NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

IMPROVEMENTS TO SOFTWARE MAINTENANCE METHODS IN REAL TIME EMBEDDED AVIATION FLIGHT SYSTEMS

by

Robert Burton Upchurch

December, 1983

Thesis Advisor: Gordon H. Bradley

Approved for public release; distribution unlimited

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER	2. GOVT ACCESSION NO. / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): Improvements to Software Maintenance Methods in Real Time Embedded Aviation Flight Systems	5. TYPE OF REPORT & PERIOD COVERED: Master's Thesis December, 1983
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Robert Burton Upchurch	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Naval Postgraduate School Monterey, California 93943	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: Naval Postgraduate School Monterey, California 93943	12. REPORT DATE: December, 1983
	13. NUMBER OF PAGES: 89
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Software maintenance, software lifecycle, aviation software maintenance, OFP documentation, OFP testing

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Software maintenance costs in Naval Aviation Operational Flight Programs (OFP) are very high and are projected to climb higher in the future. Maintenance costs are high due to poor initial design, limited programmer and system resources, poor documentation, the conditions under which the OFP must operate and the difficulty involved in performing meaningful flight software tests. The primary factors which produce the stated problems with aviation software systems are discussed. (Continued)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

S/N 0102-LF-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ABSTRACT (Continued)

The maintenance phase of the software lifecycle model proposed for standard application software systems is contrasted with that for real time, embedded, aviation software systems. A limited set of software tools and methodologies which are currently available and would greatly aid the system engineers tasked with OFP maintenance is proposed. These tools and methodologies center on two areas of flight software maintenance; documentation and testing. The thesis concludes with recommendations for future aviation flight software systems.

Approved for public release; distribution unlimited.

Improvements to Software Maintenance Methods in Real Time Embedded Aviation Flight Systems

by

Robert Burton Upchurch
Lieutenant, United States Navy
B.A. Missouri University, 1976

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 1983

ABSTRACT

Software maintenance costs in Naval Aviation Operational Flight Programs (OFP) are very high and are projected to climb higher in the future. Maintenance costs are high due to poor initial design, limited programmer and system resources, poor documentation, the conditions under which the OFP must operate and the difficulty involved in performing meaningful flight software tests. The primary factors which produce the stated problems with aviation software systems are discussed. The maintenance phase of the software lifecycle model proposed for standard application software systems is contrasted with that for real time, embedded, aviation software systems. A limited set of software tools and methodologies which are currently available and would greatly aid the system engineers tasked with OFP maintenance is proposed. These tools and methodologies center on two areas of flight software maintenance; documentation and testing. The thesis concludes with recommendations for future aviation flight software systems.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

I. INTRODUCTION

A. THE PROBLEM

Software maintenance in Naval Aviation Operational Flight Programs (OFP) has become very difficult and costly. Costs will continue to rise as new weapon systems and mission requirements are integrated into the various operational aviation platforms. Changes which reflect hardware improvements, mission changes, or improved algorithms are time consuming and can lead to long delays in delivery of the updated system. Software maintenance problems concerning the OFP are compounded by the environment in which the OFP must operate. This operational environment is a real time, limited hardware, limited support resources, and very tightly time constrained. The original design of the OFP itself was often poor and little documentation is available to the maintenance team. The OFP is written in either assembly language or a very low level programming language. Changes are made under a strict time table. Before any redesign or implementation of a change to an OFP may begin, a large effort must be expended to fully understand the OFP and the impact the proposed change may have on the entire program. Testing is a time critical task which consumes a significant amount of maintenance resources.

The maintenance effort is further complicated by a lack of trained system personnel. Personnel turnover at the software maintenance activities has been approximately ten percent per year. System programmers take on average three years to train before they able to be assigned the implementation of a significant change to an OFP. During this period the system programmer may be able to accomplish

little useful work for the maintenance activity. Due to the generally poor documentation and the complex code of most OFPs there are only a handful of people who fully understand a particular OFP. Loss of these key personnel would result in a severe reduction in the capability of the software maintenance activity to continue at acceptable production rates. There is no improvement in the availablity of competent system personnel predicted in the near future.

Due to the unique problems presented to maintenance activities by the characteristics of aviation software, maintenance costs are very significant. Fiscal 1984 operating buget for maintenance of six aircraft OFPs at Naval Weapons Center, China Lake, California, is over 16 million dollars. This figure, while seemingly high, represents only 75 percent of the requested budget. Resources are limited and the situation is not likely to improve. Several major proposed solutions have been suggested to improve the productivity and the quality of the maintenance effort. These suggestions range from complicated software development/maintenance environments to complete rewrites of the flight software itself. Budget limitations will prevent any of these major proposals being realized in the near future.

B. THESIS OUTLINE

This thesis focuses on the two areas where rapid improvement in the maintenance effort seems possible; documentation and testing. A set of software tools and methodologies which are currently available and which would have a significant impact on these problem areas are outlined. The software tools represent an affordable strategy for the maintenance activities to improve the maintenance of current flight software systems.

C. THESIS ORGANIZATION

The remaining chapters are organized in the following manner. A scenario tracing the development and maintenance of an operational aircraft system, the A-6 Intruder OFP, is presented in Chapter Two. A detailed background of the aviation software maintenance problem is given. Chapter Three gives a brief discussion of a lifecycle model for aviation software maintenance in comparision with a standard application program lifecycle model. Software tools are defined and discussed. Unfeasible solutions are explored. Chapter Four discusses software maintenance/development environments. A software development environment which is in use by the Naval Air Development Center, (NADC), Warminster, Pennsylvania, is discussed. A set of tools and methodologies which center on two areas of OFP maintenance and are felt to have the greatest impact on the productivity and quality of OFP maintenance are outlined in the next two chapters. In Chapter Five the first of these areas, documentation, is discussed. Chapter Six covers the second areas, OFP testing. The thesis concludes with suggestions for future development of OFPs.

D. RESEARCH METHODOLOGIES

1. Literature

Manual and automated searches of the literature produced a limited amount of information concerning embedded, real time computer systems. Less was found on maintenance of the software used in these systems. An automated search of government research topics dating back six years using maintenance, real time and embedded systems as keywords produced an impressive work summary. Upon closer examination, most research listed was not directly applicable to the emphasis of this thesis.

Noteworthy work dealing specifically with a Navy tactical aircraft (A-7 light attack) OFP redesign has been ongoing under the direction of the Naval Research Laboratory [Ref. 1]. In this study, not yet complete, the OFP for the A-7 was redesigned using software engineering techniques of modularity, information hiding, formal specification, abstract interfaces, and cooperating sequential processes. The study is at the point that testing both in ground simulators and flight tests is ready to commence. It will not be known if the recoded OFP will perform as required until these tests are complete. The study offers interesting insights into the problems associated with flight software systems as they are now designed.

Definitions of the critical concepts of software maintenance, software environments, and the software lifecycle are readily available in the literature. Lientz and Swanson [Ref. 2] contains an excellent definition of process of software maintenance in large application program systems. Fjeldstan, Hamlen, Bristow and Van Horn provided further definitions used for software maintenance [Ref. 3], [Ref. 5], [Ref. 4]. Guidance in the area of software environments was found in articles by Howden [Ref. 6], Bristow [Ref. 4], and Wasserman [Ref. 7]. Also the Naval Air Development Center provided an interesting discussion of their development/maintenance environment, FASP (Facility for Automated Software Production) [Ref. 8]. The model for the software lifecycle was developed from Boehm [Ref. 9]. The maintenance lifecycle was taken from Parikh and Zveginitov [Ref. 10]. The definition of a software tool was taken from work conducted by the National Bureau of Standards (NBS) [Ref. 11].

2. Laboratory Vists

A wealth of information and ideas was gathered during trips by the author to the primary Navy Flight Software Activities on the West Coast, Naval Weapons Center (NWC), China Lake, California and Pacific Missile Test Center (PMTC), Point Mugu, California. The personnel who must daily face the unenviable task of performing the maintenance on the flight software for all of the Navy attack and fighter aircraft were able to give detailed descriptions of their problems and suggestions for improvements. A tour of the facilities at both activities helped the author to gage the extent of resources available.

A conference attended by representatives from all three Navy Flight Software Labs and a group of researchers from various academic communities was held 5-7 October 1983 at the Naval Postgraduate School. Each Software Lab was given the opportunity to present what they felt were their everyday problems in dealing with flight software maintenance and their ideas for future research. The conference turned out to be both stimulating and an excellent source of information.

II. BACKGROUND

A. INTRODUCTION

This chapter traces the development and current maintenance of a typical mature flight software system, the A-6E. The primary Navy software maintenance activities are identified. The chapter concludes with discussion of the unique problems associated with real time, embedded aviation software systems.

B. A-6E FLIGHT SOFTWARE HISTORY

The A-6 Intruder is an all weather, carrier based jet powered attack aircraft built by Grumman Aerospace Corporation, Long Island, New York. Its primary mission definition is the accurate delivery of sizeable ordinance loads and close air support to ground units under all weather conditions. Since its initial design, it has taken on other roles as a carrier based tanker, electronic warfare platform and delivery vehicle for the Harpoon antiship cruise missile. Many new weapon and sensor systems have been added to the aircraft since initial production. These include laser guided munitions, Heat Seeking Antiradation Missile (HARM), Forward Looking Infrared Sighting System (FLIR), and the Harpoon Missile. It is capable of carrying both nuclear and conventional weapons. It is a subsonic aircraft operated by the Navy and the Marine Corps from both land and aircraft carrier based squadrons. The attack configuration of the aircraft is manned by a two man crew, pilot and bombadier/navigator (B/N). The aircraft was first flown in 1959 and even though the production line for the A-6 has been closed it is planned to have an operational lifespan well beyond the year 2000.

The aircraft has onboard a single, CP3 computer with 32k words of memory. The computer takes part in processing data that is involved in nearly every aspect of the operational of that aircraft. Navigation, weapon system management, weapon release solutions, radar input processing, and electronic warfare functions are all processed in some manner by the onboard flight computer. Data is input from several areas of the aircraft, processed and continuously displayed to the pilot and B/N. The computer is not necessary to fly the aircraft but without it the A-6 becomes essentially a jet powered World War Two era bomber. All major changes in weapon capabilty and mission assignment have to be in some manner incorporated into the hardware and software carried onboard. The Operational Flight Program (OFP) is the software loaded into the random access memory of the onboard aircraft computer that processes the various input and display functions.

Grumman Aerospace was responsible for the initial development, coding, integration and testing of the OFP. After acceptance of the aircraft for fleet operations, Grumman was contracted to perform all software maintenance on the OFP. This maintenance consists of removing errors found in the OFP and the incorporation of enhancements to the aircraft system into the flight software. Any change in mission definition for the aircraft must also be reflected in the OFP. Grumman held the contract for maintenance until 1978, when Naval Weapons Center (NWC), China Lake, California was tasked responsibility for all maintenance functions of the OFP. Currently most actual redesign, coding and testing of updates to the OFP are performed by personnel assigned to NWC; some work is contracted out, primarily to Grumman.

Entire OFP updates are sent to operational squadrons approximately once every year and a half. Only safety of flight or severe mission reducing software errors are given

immediate attention between scheduled OFP updates. There is a method provided for squadrons to submit desired changes and report OFP operating problems to NWC. A formal review of desired changes to the OFP is conducted by the Navy yearly with squadron and software maintenance personnel in attendance.

There are many more enhancements desired by the operational squadrons than are able to be funded for incorporation into future OFP updates. Some enhancements are not able to be adopted due to the nature of the computer system itself. The system is hardware limited. The OFP itself fills all available memory of the onboard computer. Major changes are possible only by degrading another mission area or by increasing computer performance.

C. NAVY SOFTWARE ACTIVITIES

Outlined above is the history of one Navy tactical aircraft and its flight software system. All other Navy aircraft have a similar history concerning OFP development and current maintenance. There are three primary Navy Flight Software activities. Naval Air Development Center (NADC), Warminister, Pennsylvania, is responsible for P-3C, S-3A, and LAMPS Antisubmarine mission aircraft software. Pacific Missle Test Center (PMTC), Point Mugu, California performs maintenance on the F-14A Fighter, EA-6B Electronic Warfare platform, and various missle system software. Naval Weapons Center (NWC), China Lake, California, in addition to the A-6E, has responsibility for the FA-18 Fighter/Attack, A-4M, AV-8B, A-7E, and UH1-J attack aircraft OFP maintenance. In all cases primary OFP development was done by the prime system contractor and maintenance of the software was picked up at a later date by one of the software activities listed above.

D. AVIATION SOFTWARE MAINTENANCE PROBLEMS

In the following sections the unique problems which render flight software in real time, embedded systems so difficult and costly to maintain are outlined and dicussed. Nearly all areas covered are unique to flight software and are in addition to the normal difficulties encountered in standard application program maintenance.

1. Platform

In every case the Operational Flight Programs are run on computer systems carried onboard high performance tactical aircraft. Space for hardware and support systems is limited. Primary importance is placed on aircraft weapon load and endurance capabilities. The fact that most Navy tactical aircraft are operated from aircraft carriers further defines and shapes the physical design of the aircraft. Operating an aircraft at sea subjects the airframe and internal components to severe stress during catapult launches and arrested landings. Initial design of the flight hardware system is often constrained within physical space, electrical power, and air conditioning support limitations before the hardware is selected. Once the hardware has been selected, the software is designed within hardware and mission requirements of the aircraft.

2. Aircraft Lifespan

When the A-6 was originally designed in the late 1950's the aircraft was never envisioned to have a lifespan until the end of the century. The lifespan of the aircraft will approach forty-five years. That is equivalent to a World War Two aircraft being flown today in a front line squadron. The flight software and the ability to change it to reflect new aircraft capabilities and mission

requirements allows the aircraft to remain viable for such a previously unheard of length of time. Aircraft are very expensive and as higher performance demands are placed on the newly designed airframe and tactical systems the expense will grow. The high development and purchase cost forces the Defense Department into a position in which the aircraft are utilized as long as feasible. This posture on the utilization of these aircraft well beyond their original designed lifespan has several affects on the flight software. Mission requirements and weapon systems which were never contemplated in the original aircraft and flight computer design are being incorporated into the aircraft system years later. The hardware which very well might have been state of the art during the design of the system can quickly become the limiting factor as major changes to the OFP are requested and implemented. Changes to the hardware is not an easy task and is more expensive than the high software maintenance costs. In the years since its initial design and introduction to the fleet the A-6 has undergone one major computer hardware update, while the software is undergoing constant review and change.

3. Independent Activities

High performance aircraft have a large number of very independent devices which must operate in order for the aircraft to perform its mission properly. These devices include sensors measuring various flight parameters such as altitude, air speed and angle of attack. Radar, infrared sighting, electronic warfare and weapon guidance systems, are among the many devices that flight computer systems must also react to. Input from the aircrew must be incorporated into the flight system processing as well. Interfacing these devices and inputs is a complicated task. This interface impacts greatly on the software engineer attempting to

modify a flight software system. Not only must he understand the program itself, but he also must understand the interfaces and the affect a modification will have on these interfaces. This problem is prevalent enough that managers of both Navy software activities that were visited expressed a need for system engineers rather than strict computer software engineers. It was felt that the aircraft systems are complicated enough that it is easier to train a systems engineer to program rather than train the programmer to be an aircraft system engineer.

4. Concurrent Activities

Not only are there large numbers of activities operating independently, but these activities are also concurrent in their operation. All of the interfaces with sensors and data input are constantly updated so the program can perform as required. Timing considerations in the update of these activities in critical. Input from the flight crew which is bursty in nature must be processed so that it is handled in a timely manner and does not degrade the remaining processing. Display of required information for the flight crew must be constantly updated. The display must be accurate and in real time.

5. Real Time

High performance tactical aircraft operate in very hostile conditions. Complicating the software problem is the high speed that the aircraft flys while in the hostile environment in order to enhance its survivability. This mandates that the processing of data in the flight computer system must be done in a real time manner. The definition of real time for flight systems does not equate to the definition for a banking database system. Single CPU cycles can become paramount. An aircraft traveling at 450 knots at two

hundred feet in altitude requires that updates from the onboard computer be timely indeed. A delay of milliseconds can cause the delivered weapon to miss the target entirely or loss of the aircraft itself. Every change incorporated must consider every possible affect on the timing constraints of the program.

6. Reliability and Recoverability

The degradation of one aspect of the flight software system must not allow the loss of the aircraft. The expense of the aircraft, aircrew and weapons requires high reliability in the flight software system. The system must also be able to recover from loss of input data resulting from battle damage and continue to operate in a degraded mode. The software must be protected against hardware failures as well. Failure of the entire system must only occur when the aircraft is damaged to the point of crew abandonment. Further the system cannot tolerate a requirement to restart the program due to a system fault interrupt or program crash caused by a software error.

7. Program Complexity

Due to the timing constraints placed on the OFPs, most are coded in either assembly language or a very low level programming language such as CMS-2 (P-3C) or Metaplan (F-14). The ability to perform various software engineering programming techniques commonly used in higher level languages is lost. The original design of the program is often not modular. The lack of modularity coupled with the ad hoc fashion in which changes have been made through the years has left the OFP code extremely complex. A great deal of effort is required to merely comprehend the OFP before changes are even designed much less implemented. The impact of a change to a particular piece of OFP code may have an

impact on an entirely different unrelated section of code. A case cited during one of the laboratory visits concerned a minor change to a section of code which dealt with navigation of the aircraft resulting in the inability to release any weapons. The results of changes to the code is poorly understood until the code is actually changed and testing of the revised OFP is begun. As has been well documented in the literature this a very expensive time to discover redesign errors.

The design of newer systems such as the FA-18 Fighter/Attack aircraft will show improvements in the ease of conducting software maintenance on the flight software. The A-6E has five identifiable modules which have been implemented during the last five years of maintenance by NWC, the FA-18 OFP which was written by Hughes Corporation of Long Beach, California shows a marked improvement in modularity with over one hundred identifiable modules. The situation seems to have improved much over the twenty years between the design of the A-6 and the FA-18. The FA-18 OFP is, however, coded in assembly langauge due to real time requirements of the flight software system.

8. Documentation

All Navy software activities tasked with OFP maintenance had one common complaint. That complaint centers around the lack of useful documentation received from the original designer of the flight software. While the entire subject of documentation is subject to debate as to its proper form, what is commonly turned over to the Navy from the development contractor is severely lacking. Even in the newest systems (FA-18 and AV-8B) the documentation received from the contractor has not been as extensive as the maintenance acitivity desires. Usually a program listing is the primary documentation received. Maintenance

activities find themselves not having accurate performance requirement documents on the aircraft itself or specification requirements for the OFP. Documentation carried today has largely been generated by the maintenance activity.

A problem related to documentation was identified by Neetz, [Ref. 12], of PMTC concerning the difficulty of the maintenance programmer in understanding the desired change to an OFP submitted by fleet personnel. The information contained in most deficiency reports was often found to be limited and this slowed the problem identification process. He also found that the managers felt that feedback from the fleet was adequate while the technical engineers felt it was not adequate.

9. Training

As stated earlier each software activity faces high personnel turnover. Many studies have shown that the required numbers of computer capable system engineers are not being produced. Competition with industry is keen. After a system engineer is trained adequately he may be offered a position with the contractor of the system he is trained on at a hefty salary increase. Training of a new system engineer is an extremely slow and difficult process. Adding to the problem is that often while this training period is ongoing this engineer may not be directly involved in any productive work. Since the numbers of qualified personnel is not expected to grow quickly and there is no training institute for training engineers on specific aircraft systems, all training must continue to be done internally.

10. Hardware Limitations

As stated earlier, the hardware design of the flight computer systems was often considered state of the art when

first installed in the aircraft. As the aircraft ages and more capabilities are added to the aircraft, the hardware can quickly become the limiting factor in implementing enhancements to the OFP. The A-6 was desiqnd with 16K of available RAM. Reserve memory was quickly allocated to new functions implemented in the OFP within the first few years of fleet operations. A major upgrade to 32K was accomplished in 1968, this quickly met with the same fate as the original 16K implementation. The OFP of many Navy aircraft have zero percent memory and throughput reserve. This factor leads to further complication of the OFP code when changes are made. Additions of particularly large changes to OFP code may require that certain functions of the OFP be either degraded or dropped altogether. Simply adding larger amounts of reserve memory has not been the answer due to the difficulties in making hardware changes to the aircraft itself. Also there are many more enhancements awaiting implementation that would quickly be incorporated if memory were made available.

11. Aircraft Populations

One problem facing the software maintenance activities as a whole is the limited number of aircraft of a particular type being flown. At any given time there are approximately 200 A-6 aircraft assigned to operational squadrons. The number of computers and flight programs represented by that number is not large enough to warrant large expenditures for major software redesigns and large support environments which would lower maintenance costs. Aircraft are expected to be fully supported after production lines have closed and the number of aircraft and funding support is dropping. The A-7 production line has closed but the largest change to its OFP was recently conducted by NWC when the HARM system was incorporated into the aircraft inventory.

12. Human Factors

Another obstacle facing the maintenance programmer dealing with aviation programs is the human factors associated with input and display of data for the flight crew. Human factors is defined as the functional task area which is concerned with the aspects of human performance that affect or are affected by the software [Ref. 13]. The area to which this definition refers falls primarily under the input and display of data to the flightcrew. Changes to the program which affect display are especially critical. The display must be presented in such a manner that it does not require undue effort for the flight crew to comprehend it. Little is understood in this field of computer science. Research is currently being conducted at PMTC dealing specifically with human factors as they relate to flight programs. Programmers implementing changes to an OFP must constantly keep in mind the affect of their change on any display data. Guidelines for the affect on the flight computer operators is not based on scientific fact rather it is based on operator feedback.

13. Military Standards

Currently all software maintenance activities operate under several Military Standards (Mil-Std) which guide the development and maintenance of the programs they cover. Primary in importance to the flight software systems is Military Standard 1679A (February 1983) which covers Weapon System Software Development. Unfortunately this standard while good in its intent does not address the real world situations of OFP development and maintenance. Several of the active OFPs were written ten to fifteen years prior to Mil-Std 1679 first being issued (1978). Concepts covered by Mil-Std 1679 were not applicable when these older

OFPs were designed. Mil-Std 1679A requires the use of a high level programming language in all weapon systems. As has been mentioned earlier, it is not possible to code current OFPs in a high level language due to timing constraints. NWC personnel have expressed some concern over the requirements outlined in Mil-Std 1679 and the difficulties in following it on an OFP as complex as the A-6's has become. PMTC personnel have no real complaints about it. But it should be remembered that they are working on somewhat newer systems to which it can be more readily applied. A review of Mil-Std 1679 is contained in [Ref. 14].

14. Deadlines

The software activities maintaining flight software on operational aircraft often find themselves facing severe deadline requirements. Safety of flight or primary mission degrading problems with the OFP are processed on an immediate basis requiring the possibility that all other OFP maintenance tasks be dropped. If the redesign of the OFP is not properly done and the error is not discovered until late in testing phases, nearly the entire process must be repeated and delays in OFP updates may be experienced. In order to meet strict deadlines certain update features cannot be accomplished. This constant time deadline influneces the performance of the maintenance effort throughout its cycle.

15. OFP Testing

Before a revised OFP can be considered safe to flight test extensive ground testing is completed. This testing requires massive support facilities in the form of flight simulators. The target aircraft computer is loaded with the revised OFP. The support facilities suuround and interface with the target system suppling input data to

exercise the OFP. The support facility measures the performance of the OFP under the simulated flight conditions. Because of the complexity of the OFP code, poor documentation, and high reliability required, testing is the most expensive of the operations performed on the OFP during a maintenance change. Little is known on the exact method to test the code to yield meaningful test results. The nature of the OFP code itself and the mission it must perform renders the testing of the code even more difficult than normal.

16. Scope of Maintenance Changes

Industrial application programs normally face a five percent per year code growth due to software maintenance. The maintenance of OFPs produces something on the order of twenty five percent code change per OFP update. The shear amount of code required to make the changes during an update cycle contributes to the difficulty of the maintenance. After the completion of two to three OFP updates the code may be significantly changed from the orginal program. If not well documented, the high volume of maintenance changes will render the program almost unfathomable. A problem faced by the maintenance personnel is that the code has already gone through several updates prior to being turned over to the Navy.

III. MAINTENANCE IN NAVAL AVIATION FLIGHT SOFTWARE

A. INTRODUCTION

The question of why software is the important product in aviation flight computer systems is addressed first. A general description of the software lifecycle and the maintenance lifecycle as related to aviation systems is given. Proposed solutions to the flight software maintenance problem are given. The following definitions are outlined: software tool, software maintenance, lifecycle model, and maintenance lifecycle model.

B. WHY SOFTWARE?

When first designed and built, real time embedded computer systems had their functional capabilities primarily embodied in the electronics with software playing a minor role controlling ancillary functions. Demand on the performance of these systems required that they be designed with a greater degree of inter-system communication between devices. This has caused the software of these systems to shift from a minor role to one where the system functional definition is in the software and the electronics are only a means of providing for execution.

Boehm [Ref. 9], defines software as the entire set of programs, procedures and related documentation associated with a system. The Software Technology for Reliable Systems (STARS) Program Strategy Handbook lists in addition: definitions, designs, testing materials and maintenance instructions [Ref. 13]. Software is what controls the computer and allows it to accomplish so much. The hardware in the actual computer systems of the tactical aircraft undergoes few

changes throughout the lifespan of the aircraft. Yet the flight software system is expected to be constantly upgraded as additions and enhancements to the aircraft system are implemented. These changes are primarily reflected in software.

The U.S. Air Force experienced a situation that illustrates the case for software in embedded computer systems. F-111 tactical aircraft were operated in two basic models. In one, avionic systems were implemented in analog devices while in the other the same systems were implemented in digital devices. The Air Force was tasked to keep the capabilities of both models equal. Several changes to the systems were tracked and it was determined that changing the hardware implementations was roughly fifty times as costly as the software changes [Ref. 13]. The cost and time to design a software change is roughly equal in cost and time to design a hardware change. Hardware however, requires management of individual changes and physical copies of the new hardware be maintained. Software is much easier to copy on multiple tapes and quickly load into the individual computers. The difficulties in implementing changes with hardware are evident when compared to implementing the same changes in software.

PMTC personnel point out the case of the F-14 as another example of why improvements to the aircraft computer system are best carried out in software. F-14 OFP changes are promulgated approximately every two years at a total cost of roughly two million dollars for each change development and implementation. There are approximately 400 F-14 aircraft. Implementing the changed OFP in each aircraft consists of merely loading the new OFP tape into the aircraft computer system memory. The cost of a new OFP is approximately 5,000 dollars per aircraft. Anyone experienced in making equipment alterations to military aircraft knows 5,000 dollars

will buy very little. When considered against the cost of an individual F-14 (@ $30 million) implementing flight computer system changes through software is very cheap. If all of the corrections and enhancements to the system had been made in hardware the costs would have been in the billions of dollars.

C. SOFTWARE MAINTENANCE

Fjeldstad and Hamlen define software maintenance to be incorporation of changes to existing programs, using or modifying an existing approach or design then understanding and modifying or expanding existing program logic [Ref. 3]. Lientz and Swanson describe the primary types of maintenance [Ref. 2].

1. Corrective Maintenance: correction of errors introduced in the software through improper logic or coding errors.

2. Adaptive Maintenance: satisfaction of changes in processing environment. Input and output requirements often change. This case was experienced with the A-6E system when the aircraft's navigational suite was upgraded.

3. Perfective Maintenance: enhancement of the system for increased performance and maintainability. This includes improvements to documentation and recoding to improve program efficiency. Again using the A-6E as an example, the aircraft was tasked to perform low level bombing from two hundred feet vice five hundred feet. This change was induced to increase weapon accuracy while also increasing aircraft survivability against heavily defended targets. This improvement in the aircraft mission definition required extensive OFP software modification.

Lientz and Swanson in [Ref. 2] offer the following statistics on the allocation of maintenance time. Twenty percent of the maintenance effort involved corrective maintenance. Adaptive maintenance accounted for twenty five percent. Perfective maintenance accounted for the rest of the time at fifty five percent. Enhancements accounted for the largest share of the perfective maintenance at forty nine percent of the total maintenance effort. These figures

were taken from a survey conducted of large data processing organizations.

Results of an informal survey of NWC maintenance time yielded slightly different figures. Corrective maintenance of errors which are present from the last OFP update or earlier, only comprise five percent of the maintenance time. Adaptive maintenance is roughly the same as the Lientz findings at twenty percent. The largest share of the maintenance time in OFPs resides in perfective maintenance. This involves mainly optimizing the code and incorporation of enhancements to the aircraft system as a whole.

Van Horn defines another form of software maintenance, that of restructuring, [Ref. 5]. Restructuring involves change to the internal structure of the program while not changing the overall external behavior. This is interestingly a consideration for improvement to many of the older OFPs and was implemented by the Naval Research Laboratory [Ref. 1] for the A-7 OFP.

D. SOFTWARE LIFECYCLE

Figure 3.1 presents the standard waterfall software lifecycle as seen in Boehm [Ref. 9]. This model represents the development of a standard large scale application software system. It is based on two assumptions:

1. Each phase of the lifecycle is culminated by a verification phase that attempts to eliminate errors in the output of that phase. This is expected to be accomplished prior to moving to the next phase.

2. Iterations of earlier phase products are performed in the next succeeding phase.

Each phase of the Boehm Lifecycle Model is briefly described below:

A. Feasibility: Defining objective of the proposed software product. Is it feasible to be accomplished? And will it be superior to the system that it is proposed to replace?

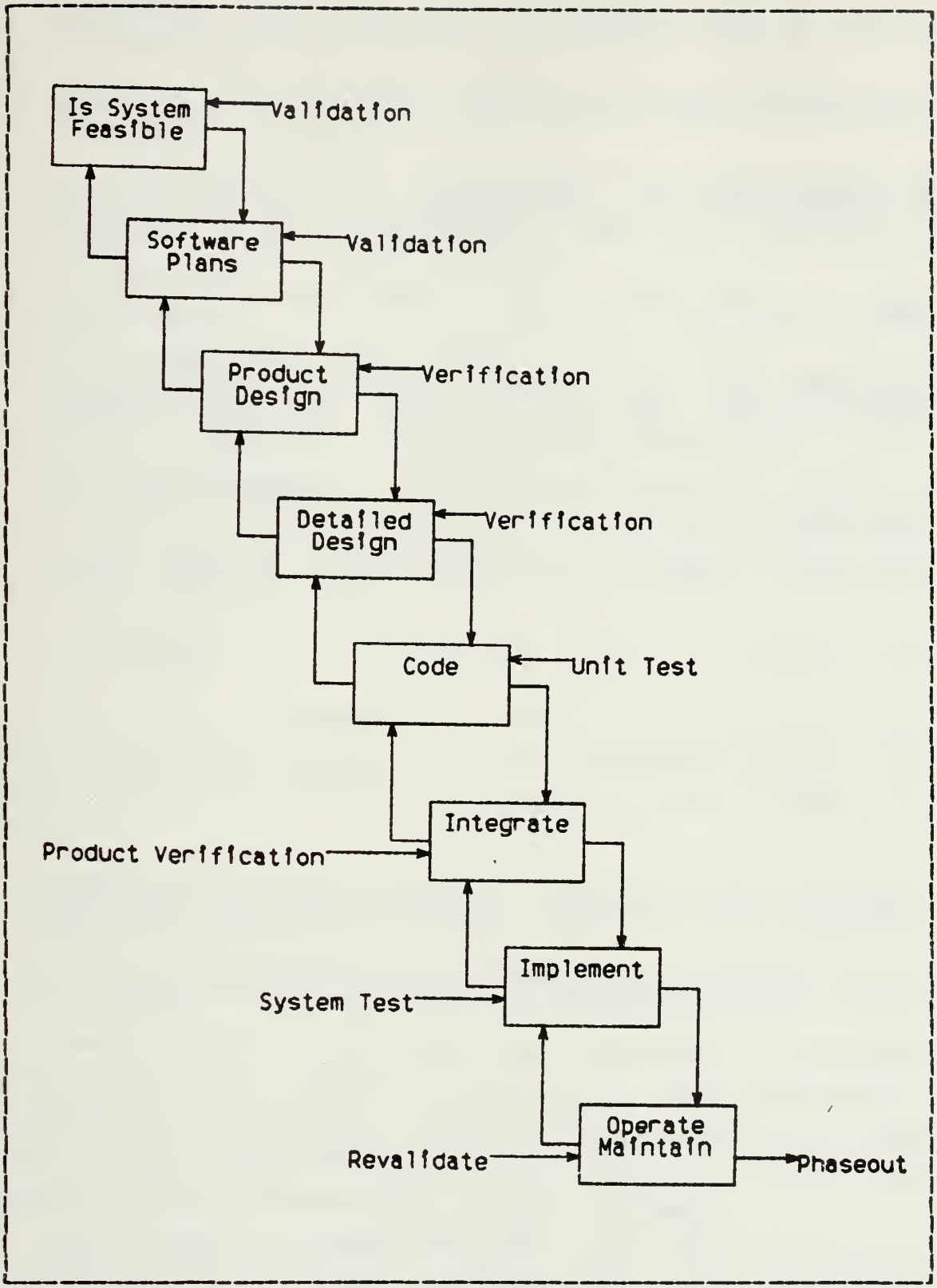

Figure 3.1 Boehm Software Lifecycle Model.

B. Requirements: A validated specification of required functions, interfaces and performance aspects of the proposed system is generated.

C. Design: The high level hardware-software architectural design, control structure and major data structures for the system are outlined.

D. Detailed Design: Complete verified specification of the high level design is produced. Precise algorithms, data structures, interfaces and control structures are designed. Several refinement steps are involved as detail of system is realized.

E. Code: The software portion of the system is implemented in executable code. Testing of individual components begins.

F. Integration: The software product is made functional and is run. Individual components are integrated into subassemblies and finally into the final software product. Initial errors are removed from software as they are identified. Program testing continues.

G. Implementation: The software-hardware system is brought into initial operation. Testing is completed to determine if the overall product meets design objectives.

H. Maintenance: Error corrections are made to the operational program. Perfective and adaptive changes are accomplished as needed.

I. Phaseout: A replacement system is designed and implemented.

The system is sequential in nature and the start of any phase assumes the completion of the proceeding phase. The verification and validation part of each phase is defined as follows:

Verification is the process by which the truth of correspondence between the software itself and its specification is assertained.

Validation establishes the fitness of the software system in carrying out its intended operational mission.

Boehm further states that the lifecycle as proposed allows for a high degree of control in the configuration management of the product. The manager is able at any given time in the development/maintenance process to define the specific state of the project in concrete terms.

Once a design strategy following the lifecycle model is implemented the project baseline can be established.

According to Boehm the three major advantages gained from this baseline are:

1. No changes are made to the system without agreement of all interested parties.

2. Higher threshold for changes will stabilize the product.

3. The overall manager controls the configuration management process.

The lifecycle model as presented by Boehm is a well known and accepted model. The question remains just how well does the model comply with real life systems. When comparing this model to the development and maintenance lifecycle of a typical aviation software system it seems not to compare well at all. The current method of operation in OFP maintenance has the maintenance activity stepping into the lifecycle model at the next to the last phase. The software activities have in the past had little input into the software development process conducted by the prime system contractor. There is little or no communication in the form of documentation when the software activity assumes responsibility for maintenance. The logic and design methodologies used by the original designers are lost to the maintenance personnel. The continuum of the lifecycle model as proposed by Boehm is lost when the Navy begins maintenance of the OFP.

Boehm also gives little mention of the maintenance phase itself in his discussion of the lifecycle model. Several studies have found that the maintenance phase of the lifecycle the most expensive. Estimates range from fifty to eighty percent of overall system costs are involved in software maintenance of a large application software system [Ref. 2]. A U.S. Air Force study estimated that software costs during developemnt averaged $75 per instruction. During the maintenance of an operational system the software

costs increased to $4000 per instruction [Ref. 15]. This trend is reflected in aviation flight software systems as the lifespans for the aircraft they serve are extended.

E. MAINTENANCE LIFECYCLE

The maintenance phase can be thought of as a lifecycle within the overall lifecycle. Incorporating enhancements to a system or repairing errors not found during initial testing phases will involve a redesign effort similar in many respects to the initial design of a system. Parikh and Zvegivtov [Ref. 10], review a maintenance process in the opening comments to a chapter in their book on Software Maintenance. Figure 3.2 illustrates this process. This representation is based on changes being made to a fully operational system. Their simplified maintenance lifecycle is defined as follows:

1. Understand the Request: The user of the system requests a change to an operational system be made by the maintenance activity. This request is written in a langauge familar to the user. The maintenance programmer must understand the request and the current program prior to design of the change.

2. Transform Request: Using a description of the existing and requested systems, the differences between the two are sought. The process of designing for the change involves reducing the differences between the existing and the new system. The existing system is revised to match the new system.

3. Specify Change: A Cut Line and Patch are specified. The Cut Line is the section of code to be modified. The Patch is defined as the new code to be implemented which reflects the new system within the Cut Line. The selection of the Cut Line is difficult because it is selected to minimize interaction between the existing system and the Patch. Lowering this interaction will reduce the chances of damage to other sections of the program not necessarily related to the Cut Line. Modularity of the program is a key aid in selection of the Cut Line. Program complexity is another issue which affects the interaction of the Patch once inserted within the Cut Line.

4. Develop the Patch: The Patch is actually developed in a programming langauge using standard development techniques. The ultimate goal of the Patch is the accomplishment of the requested change to the existing program. The Patch should be designed such that it will fit within the Cut Line.

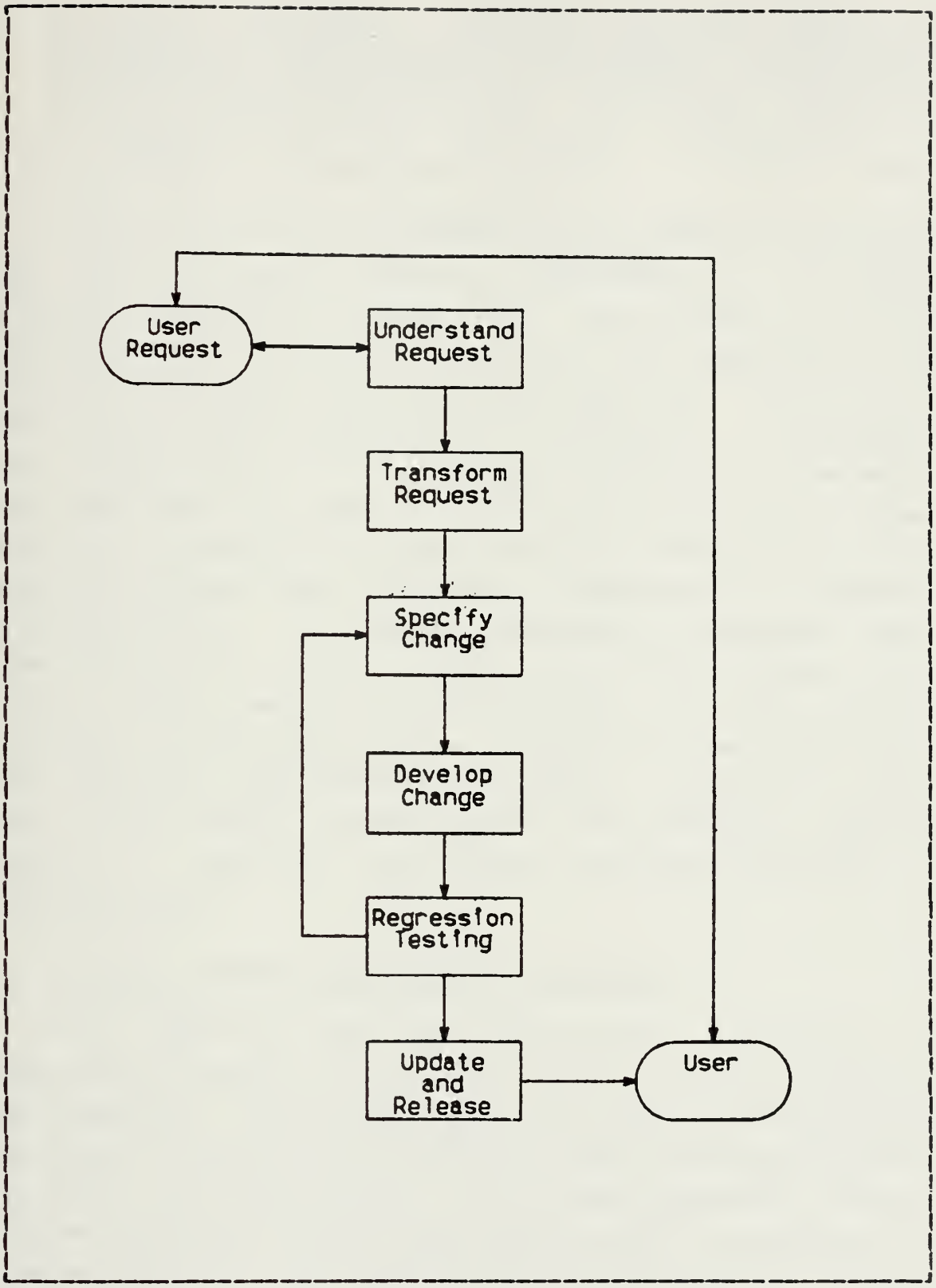

Figure 3.2 Parikh and Zvegintov Maintenance Lifecycle.

5. Test: The change is installed and tested within the development environment. The Cut Line is tested for appropriate switching between the existing functions and the new code. The impact of the new code to code outside of the Cut Line is identified. Regression testing is performed where needed.

6. Release: Once tests are performed the updated system is installed and released.

The model for the maintenance cycle presented by Parikh and Zvegivtov can be seen as a refinement of the overall lifecycle presented by Boehm. Its major concern is in the understanding of the request, relating that knowledge to the existing system and designing the change such that it will not degrade the updated system. Interaction is addressed more specifically. It is perhaps optimistic in assuming that the Patch can always be installed within the Cut Line. Also the selection of the Cut Line while difficult in a well designed program would seem nearly impossible in a complex software system. The rather difficult and extremely large area of testing is given a quick review in this model. The cycle assumes several characteristics of the existing program, such as proper design, adequate documentation and a well developed maintenance environment. This may render this model somewhat simplified for the embedded computer system. A model for the aviation software lifecycle is presented next.

F. AVIATION SOFTWARE MAINTENANCE LIFECYCLE

The aviation development/maintenance lifecycle as presented in Figure 3.3 was taken from a slide presented by NWC personnel. Further discussion was held with maintenance personnel to determine how close the real maintenance effort parallels this definition. The model presented roughly follows the Boehm model. The aviation model is presented in greater detail. Each phase is well documented by required submissions of reports and specifications. Reviews, audits

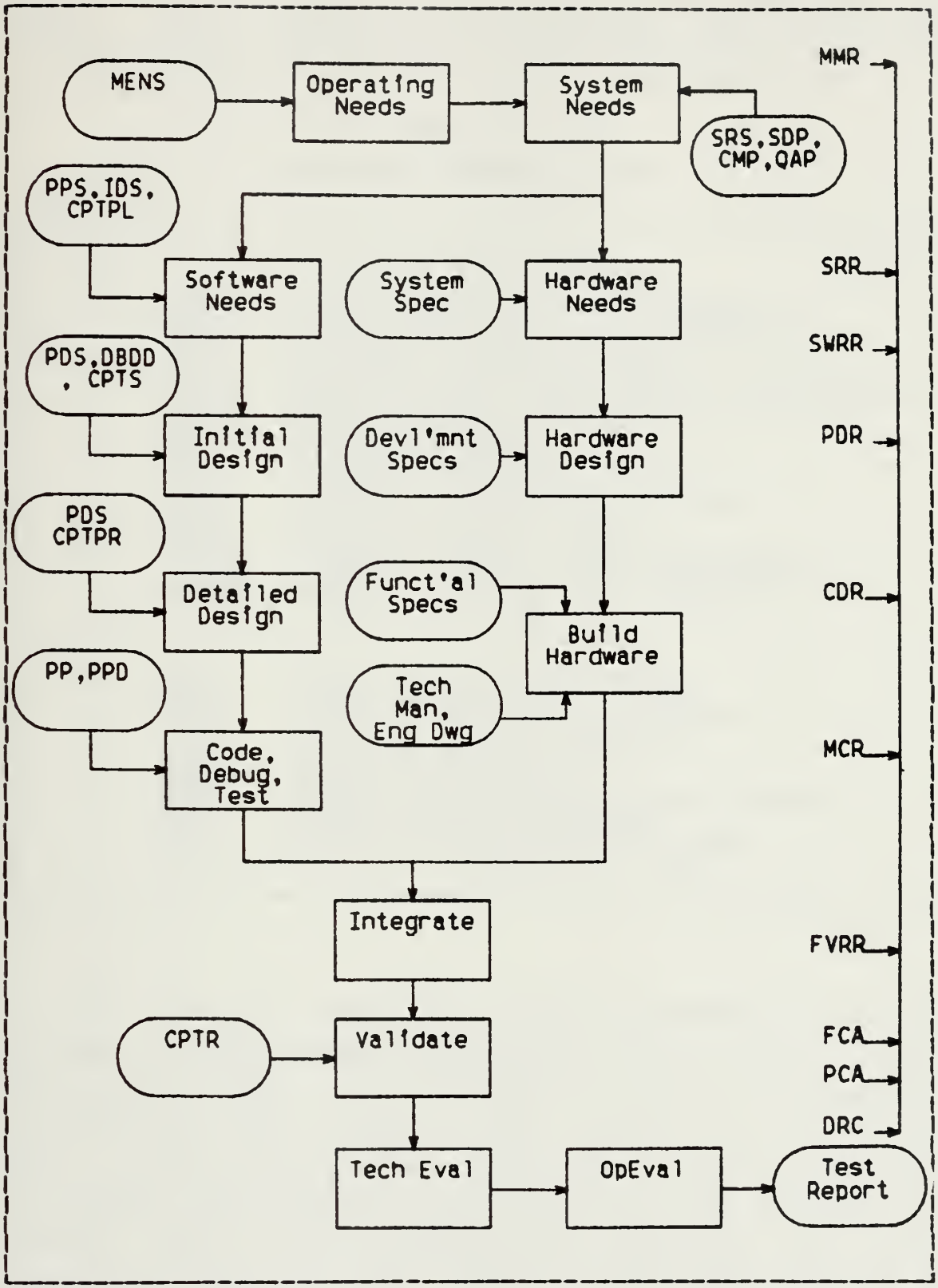

Figure 3.3 OFP Development/Maintenance Lifecycle.

and walkthroughs are scheduled at several points of the model. Table I defines the abbreviations used to represent the documentation and reviews shown in the model.

TABLE I

Aviation Lifecycle Terms

	Lifecycle Documentation
MENS:	Mission Element Need Statement
SRS:	System Requirements Specification
PPS:	Program Performance Specification
IDS:	Interface Design Document
PDS1:	Preliminary Program Design Specification
PDS2:	Program Design Specification (Final)
DBDD:	Data Base Design Document
PP:	Program Package
PDD:	Program Description Document
SDP:	Software Development Plan
CMP:	Configuration Management Plan
QAP:	Quality Assurance Plan
CPTPL:	Computer Program Test Plan
CPTS:	Computer Program Test Specification
CPTPR:	Computer Program Test Procedures
CPTR:	Computer Program Test Report
	Reviews and Walkthroughs
MMR:	Mission Requirement Review
SRR:	System Requirements Review
SWRR:	Software Requirements Review
PDR:	Preliminary Design Review
CDR:	Critical Design Review
MCR:	Module Code Review
FVRR:	Formal Validation Readiness Review
DRC:	Design Review Committee
FCA:	Functional Configuration Audit
PCA:	Physical Configuration Audit

The model as presented is well structured and well defined. Unfortunately the reality of what actually occurs during development of maintenance code may not be accurately represented by this model. There are several reasons for this. In many of the older flight systems the exact process of software maintenance was not precise and was difficult to define. Some of the documentations specified and reviews presented in the model are currently not being conducted in

every flight software system. The model presented represents a standard that several of the OFP maintenance teams are attempting to achieve. What is actually occuring is only partially represented by this model.

Nearly all flight systems undergo few hardware changes throughout the lifecycle. The hardware branch of the model only applies to new development of an entire flight system or a major midlife modification. The year to year software maintenance of the OFP has the software branch of the model taking on the most impact. New additions to the hardware and complete changes are usually done with hardware already in use in other systems. This also significantly reduces the amount of time spent in the hardware branch of the lifecycle.

The integration phase of the aviation model represents primarily the integration of the old and new OFP code. The complexity of older system code makes this much more difficult than the integration of the entire OFP with the system hardware. Also code may be developed by parallel development activities. Contractors are often utilized to perform the generation of portions of maintenance code. Since these parallel redesign efforts are usually conducted on different systems, conversion of the code developed outside may be required in order for it to be integrated and tested at the Navy software facility.

One high level manager involved in the maintenance effort on one of the older OFPs told of the reluctance of the system engineers and programmers to adopt any standard model for the maintenance process. They are used to doing it a certain way that is comfortable to each indiviual programmer. A standard is difficult for them to accept. The stages are well documented. The reviews and walkthroughs require the programmer to spend a great deal of time preparing for them. One programmer complained of

spending over twenty hours preparing for a review on a small section of OFP code which required one hour of time to recode.

Even though it may not be exactly standardized for each maintenance team, the model presented in Figure 3.3 presents a good general representaion of what each OFP undergoes at one time or another during development and maintenance.

G. SOFTWARE TOOLS

1. Definition

Software tools are defined by the National Bureau of Standards [Ref. 11], as computer programs that aid in the specification, construction, testing, analysis, management, documentation and maintenance of other computer programs. Shooman divides these many functions into four broad catagories, [Ref. 16].

a. Program editing and storage
b. Program processors and preprocessors
c. Program configuration and control
d. Testing and debugging processes

The purpose of a Software Tool is to aid the programmer in such a manner that productivity and the product quality are increased. They are designed to be used many times on several different projects within several different environments.

In [Ref. 7] Wasserman gives the attributes of a useful tool as the following:

1. Singularity of Purpose: The tool should be designed for one primary use, carrying out one well defined function.

2. Ease of Use: The tool should not burden the user. The programmer should want to use the tool to increase his productivity.

3. Self Documenting: The tool should not have large hard copy documentation but instead most documentation should be in the form of an interactive help facility.

4. Consistency: Each tool should be consistent with the others of the environment in which they are contained. The product of a tool used earlier in the lifecycle of the software should be able to be used by another tool used in a later phase. To achieve this tools should interact through common interfaces. Tools within each environment conform to a set of standards so that familarity with one tool will help in learning another tool.

5. Adaptability: A tool should be able to adapted to some user desires. The tool should have several modes available from a basic generic mode up through the full design capability of the tool.

6. Local Intelligence: The tool is able to capture useful data from the environment in which it is employed. Normally this is stored to a data base where it may be further processed for documentation and configuration management purposes.

2. Software Tool Usage

There seems to exist a general agreement within the literature that well designed software tools are highly desirable. Precise tool definition and terminologies are not well defined. In [Ref. 11], a taxonomy of software tool definitions and terminologies are standardized in order to allow comparision among different tools. Software tools are large computer programs, which like any other programs, face the same development and maintenance problems. They are expensive to develop and may not always meet the original specifications.

Other than expense there are several other reasons that software tools have not found wider use. Nassi [Ref. 17], lists several general catagories which have hindered the use of software tools.

a. General Nature of Many Tools

Some tools are very general in their intended use and are not at all suitable in some specific systems without a large modification effort. To develop a specific tool for a specific application may not be worth the

development costs when compared to the savings it will generate. This is a common case for lack of tool use in the aviation software field. Some aircraft computer system populations are low and do not warrant the expensive that development of a tool for that particular OFP would entail.

b. Learning Curve

Programmers accustomed to working in a certain environment may find the pain of learning a new tool not worth their effort. Even if the programmer can be shown to benefit greatly from the use of a new tool, habit may make the adoption of that tool difficult. A tool which is particularly hard to use or learn is doomed to failure. Usability of the tool should be such that the programmer is not encumbered by its use. It should compliment the environment in which it is used , not fight it.

c. Functionality

If a tool is not suitable for a specfic job it may create performance burdens on the system on which it is being used. The overhead created by its use should not be excessive. The tool should be reliable in that it may often be operating directly on user source files and the programmer must be able to trust in its use.

d. Integration

The integration within an environment should allow for the tool in use to communicate easily with other tools. The programmer will then be able to move smoothly back and forth between stages as needed without a great deal of effort.

e. Tool Usage by Software Activities

Coupled with the reasons cited by Nassi for the limited use of software tools, the flight software maintenance activities face other problems. Funding to purchase the tools is not available. The software activities conducting maintenance on the OFPs do use a varirty of software tools. Most seem to be generated in house for specific purposes within the enviroment of a particular OFP. They are often not portable to another project. The use of more powerful off-the-shelf tools has also been hindered by several factors. The state of most OFPs currently would require extensive reconfiguration to allow the use of these tools. The worst problem is the lack of documentation. Many tools developed by industry require a well designed, well documented program to work with. An example is the TRW developed software tool SREM (Software Requirements Engineering Methodology). SREM requires that documentation in the form of an adequate set of program requirements be available. Most OFPs do not have such documentation and thus cannot use SREM in the production of flight software code.

The environment of the maintenance effort for the various OFPs differ from project to project. None of them seem to be able to support a set of tools which would cooperate and communicate during the maintenance process. The work required to set up the environment and program to work with an off-the-shelf tool has been found in many cases to be excessive. Internal development of powerful tools is also time consuming and may not be feasible.

The Av-8/A-4 test facilty at NWC conducted a survey of all software tools in use in their facility. The results are interesting and reflect the situation throughout most OFP maintenance and test facilities. Forty four

different tools were listed as in use. Ninty five percent of the tools were developed internally by personnel assigned to the test facility. Twenty nine percent have the ability to communicate with one or more tools. Fifty percent of the tools had no support available. It is easy to see that this is a long way from the ideal situation many authors propose for automated tool usage.

The maintenance activities find themselves unable to buy their way out of the OFP maintenance problem by designing or buying tools. Once a software system is accepted from the developer the original design and documentation may limit what the maintenance activity has available to improve the maintenance effort.

H. PROPOSED SOLUTIONS

Many solutions have been proposed to ease the software maintenance problem in common application systems. A smaller list of solutions have been proposed for embedded systems. Solutions range from the incorporation of good software engineering practices in the design of the software to the use of extensive programming environments throughout the lifecycle of the program. Most of these solutions are viable and would help if they were to be applied from the original design of the software. Several of the proposed solutions are outlined. The reasons for the nonuse of these solutions are also cited.

1. OFP Rewrite

Many of the flight software systems still in use were designed before many of the software engineering practices that are today taken for granted were in common use. A complete rewrite of an OFP using these techniques has been suggested as a possible solution to the maintenance problem.

This was the idea behind the work of the Naval Research Laboratory [Ref. 1], in the recoding of the A-7 OFP. The use of the software engineering techniques of modularity, information hiding, formal specification, abstract interfaces and cooperating sequential processes were used in the updated OFP as it was rewritten. It is hoped that these techniques will lead to lower maintenance costs of the OFP.

An entire rewrite of the OFP offers several clear advantages. It is in fact considered the only method to assure lowering of maintenance costs. Many maintenance personnel interviewed about solutions to the OFP maintenance problem mentioned OFP rewrite as the best method to show the most improvement.

Rewriting the OFP in either assembly language or a suitable high order language would allow generation of currently nonexisting documentation. The A-7 rewrite has produced a well documented program. A significant finding of the A-7 rewrite work was the the importance of a Software Requirements Document. Its generation for the A-7 was very time consuming. It might be as important to the maintenance function as the modern software engineering principles used in the rewritten OFP. The production of documentation in a usable form would have significant impact on training of system personnel as well as the actual maintenance of the program. The documentation could also be designed with the eventual use of more extensive and supportive software tools in mind.

The program itself would be placed into a more maintainable state by incorporation of modern software engineering techniques into the redesigned code. This was the ultimate goal of the NRL work on A-7. It is very easy to see conversion from the "spaghetti code" that many of the OFPs contain to a modularized format would have great impact on the reduction of maintenance costs. The modern version

of the OFP would also be easier to test with the reduction in the complexity of the code. The given state of the program may be every bit as difficult to determine due to the complex nature of the platform and mission of the program itself but errors would much easier to isolate once detected.

Several problems do exist in this solution. Costs to accomplish a rewrite are very high. A complete rewrite of the A-6 OFP is estimated by NWC personnel to cost upwards of 20 million dollars and take four years to complete. The finished product would reflect the state of the OFP when the rewrite was begun. The ongoing enhancements occuring in the operational OFP would still have to be incorporated in the rewritten OFP. Meanwhile the existing OFP would have to be continually maintained as is currently practiced. The estimated A-6 OFP rewrite cost represents more money than the entire yearly operating budget of the software laboratory at NWC. The cost in time and the personnel required to accomplish the project may in fact be the determining factor. The personnel are not available to accomplish the rewrite and carry on normal maintenance activities of the operational OFP.

The lifespan of the aircraft in a particular modification is subject to change. The days of the A-6E system may be numbered. An F-model is under consideration which would represent a complete change in many of the systems from the E-model. The future of the F-model is in the hands of Congress. When the F-model will come on line and work slowed on the E-model is unknown. If the funds were available to rewrite the A-6E OFP it is hard to imagine them used to actually begin recoding work with the possibility of a major avionics and flight software modification arising in the A-6F model.

Questions remain about the final product of such a rewrite. Naval Research Laboratory has not yet completed its work on the A-7 OFP rewrite four years after the original completion date has past. If the new OFP will fit the available hardware in the aircraft and perform as the old OFP, remains to be seen until after testing phases are completed.

For the reasons outlined above, a complete rewrite of existing OFPs is not feasible at this time.

2. High Order Languages

A rewrite of an OFP is usually suggested to be accomplished in a standard Navy approved high order language (HOL). Experience with OFP maintenance by the various software activities has shown that the use of a HOL may not really be required. NWC personnel estimate that very little of the time spent on the maintenance of the OFP is spent in actual coding of the maintenance change. Ignoring the software engineering techniques that a true high order language affords, very liitle is gained by recoding the OFP. The ability to modularize an assembly language version, complete with documentation on each module would be as useful. The use of a high order language also presents the problem of execution speed. The number of systems in use is not high enough to warrant spending the funds needed to write optimizing compilers to insure proper performance of the OFP. The use of a high order language would be much more suited to a system designed from the start for its use.

Some interesting ideas arise from the use of HOLs in OFPs. While perhaps not suitable for the coding of the actual flight system OFPs at this time, HOLs have been used in documentation. A HOL version of the OFP is used to help the programmer gain a grasp of what the program is actually doing prior to attempting to understand the very complex assembly language version.

A recent development is the U.S. Air Force decision to recode the F-111 OFPs in a HOL. The Air Force plans to use Jovial in this conversion. Total costs for the entire project which includes conversion of all remaining aircraft to digital avionic systems is placed at 1.1 billion dollars. Jovial is considered suited well enough for embedded applications that the Air Force feels the money required for the conversion to a HOL written OFP is worth the expense.

3. Extensive Environments

Another method to improve the maintenance effort of a software project is to improve the techniques used in its design and implementation. An improvement in these can easily be accomplished through the use of an extensive programming development and maintenance environment. The environment would be used throughout the software lifecycle of the project. This is a fine solution for new systems but hardly the answer for mature systems such as A-6 and A-7. Cost is the primary problem. Howden [Ref. 6], outlines four environments of increasing capablities and costs. The highest capabilty reflected in his proposal was designed for embedded real time systems. He estimates the capital costs at three million dollars. NADC experience with FASP, outlined in the next chapter, suggests that this figure may indeed be very low. Costs for the physical environment itself do not incorporate the modifications to a program not orginally designed for use with that environment. The modifications required may involve effort equal in cost to an entire rewrite.

4. Adding Hardware

Personnel not familar with OFP maintenance see the hardware as the primary cause of OFP maintenance problems. They feel that the maintenance problem can be solved by

addition of hardware capability through added memory and increased processor speeds. If significant increases in memory space are installed, the program may be able to be partitioned and slightly restructured to reduce complexity and decrease maintenance costs. While it is well known that costs of hardware have dropped significantly with improved technology and the costs of software continue to rise, additions of large amounts of memory or increased processing speed is not the answer to the maintenance problem. Physically placing new or additional hardware into the aircraft is very difficult and requires extensive study before approval. Due to the long process to research and approve changes to the aircraft itself, addition of even a small hardware change becomes quite expensive. The older aircraft support systems may not be compatible with some of the newer hardware technologies rendering the addition of the new hardware even more difficult and costly. When memory additions have been made in the past, many new capabilities and weapon systems are added which quickly fills any newly available memory space. Merely throwing hardware at the problem is not a solution.

IV. SOFTWARE ENVIRONMENTS AND FASP

A. INTRODUCTION

This chapter will briefly outline the concept of software environments and review the NADC operated Facility for Automated Software Production (FASP), the only current attempt at a complete development and maintenance environment.

B. ENVIRONMENT DEFINITION

The concept of a Software Engineering Development and Maintenance Environment is outlined in [Ref. 7]. The environment is generaly defined as the technical and management methodologies, the hardware, mode of computer use, automated support facilities (tools) and the actual physical workspace. It encompasses every aspect of the development and support of a software system. The ideal environment should support a development methodology. Wasserman states that this has not generally been the case in many past efforts. It also should support the software system throughout the entire software lifecycle. A specfic definition of the lifecycle should be incorporated into the design of the environment. The STARS Program Strategy Handbook gives a broader definition of an environment to include the personnel assigned to use the environment.

A complete development and maintenance environment should possess the following characteristics:

1. Complete Lifecycle Coverage: The methodology supported by the environment should cover the entire lifecycle. A means for software system design is followed by a method for the code design and implementation. The environment also supports the software system through the maintenance phase.

2. Ease of Transition Between Phases: Building on the support of the lifecycle, each phase within the lifecycle should be able to be identified and traversed by methodologies employed by the environment. The transition should be painless and allow the programmer to move backwards as needed to correct or change earlier work.

3. Ease of Use: The environment should be designed such that the programmer is not burdened by its use. The personnel assigned to the project should be able to learn the environment's methodologies without undue effort. The training of new personnel would be made easier, allowing them to become productive members of the teams more quickly.

4. Repeatability: An ideal environment is general enough to be used several times on functionally similar but different actual projects. The effort in creating a complete environment tailored only to a specific system is lost when that system is no longer in use.

5. Automated Support: Since the ultimate goal of an environment is the increased productivity and quality of the product, the selection of the automated support facilities is critical. The tools selected are automated to the extent that an increase in productivity is gained through their use.

Boehm cites a study in which the COCOMO Model for Software Cost Estimation was used to demonstrate the effectiveness of the use of a properly designed environment. Figure 4.1 shows the estimated improvements in software productivity versus software cost driver attributes. From the graph presented in Figure 4.1 several of the software cost driver attributes can be seen to impact greatly on the flight software problem as defined in Chapter Two of this thesis. Most notably, schedule constraints, turnaround time, software tools, storage constraint, required reliability, program complexity and personnel capability greatly influence the maintenance effort of OFPs. Many of these factors are out of direct control of the maintenance personnel due to the nature of the flight programs and the development practices used. It appears from the data presented by Boehm that concentration in the areas of increasing personnel capability through tools and methodologies will have the greatest impact on increasing maintenance productivity. Interestingly, testing problems are not addressed directly by Boehm as a Software Cost Driver Attribute.

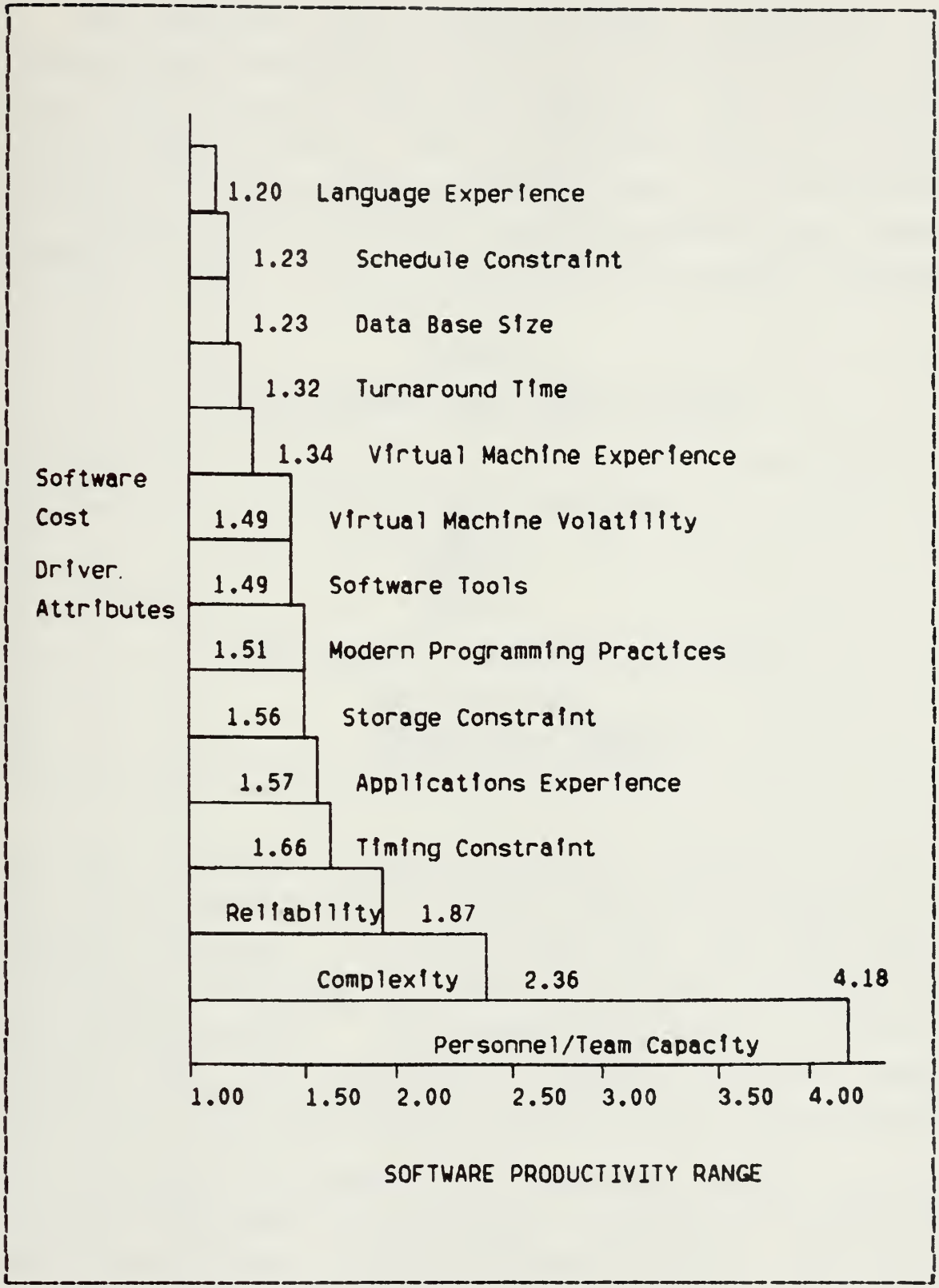

Figure 4.1 Multiplicative Software Productivity Factors.

C. FASP

FASP (Facility for Automated Software Production) was designed and implemented by NADC in recognition of the high costs and complex nature of developing and maintaining weapon system software. FASP is currently used in the maintenance of antisubmarine aircraft software. It was designed to be used in the development and maintenance of any weapon software system. Table II gives the Navy standard computers

TABLE II
FASP Language and Computer Support

Navy Standard Computers
AYK-14
AYK-10
UKY-20
UKY-7
UKY-32
Navy Standard Programming Languages
SPL/1 and SPL
CMS-2M and CMS-2Y
MACRO-20 and ULTRA-32
FORTRAN and COMPASS

and Navy standard programming languages supported by FASP. The total lifecycle of the software system is intended to be supported by FASP. It was designed such that the primary development contractors are able to use FASP throughout the development process. The maintenance activity is able to inherit from the contractor a complete software system developed on the same support facility it will be maintained on.

Two types of facilities are provided through the FASP system. The first is for software integration. Integration facilities consist of laboratory simulations of the target

aircraft computer system. The integration facility is used in hardware/software integration. It serves as the hardware configuration baseline and is also used in the determination of the human factors involved in system design and maintenance. Change proposals to a software system can be quickly evaluated by use of the simulated target computers.

The software production facility, the second facility provided by FASP, uses an approach to software development in which the same facilities are used for both development and maintenance. The software production facility was designed to be shared by several software systems for their entire lifecycles. Improved software tools provided by FASP increase programmer productivity and product quality. Management visibility of the software configuration is provided. Maintainability is increased through the support of structured programming and modularity techniques.

An integrated database which contains project and management data is ultilized extensively. Maintenance and development is divided within the database into distinct processes each with a measurable output. Input and output of each phase is stored in the database where it is automatically configured into management reports for each project. The project manager is able to set production figures into the database which FASP will automatically track and report on. The configuration control provided by FASP allows more accurate cost estimates on software production.

Automation provided by FASP reduces the production effort in the labor intensive areas of development and maintenance. Increased programmer productivity will offset increasing programmer costs by decreasing computer time required in these areas. FASP performs the following automatic operations:

1. Translation of simple user commands into many operating system commands
2. Maintenance of the database.

3. Execution of regression testing on specific software modules and report of test results.

4. Interactive program editing and testing

These automatic features free the programmer from many routine tasks and allows greater use of program librarians.

FASP provides a formalized structure which contains the software tools necessary to increase production and quality of the final product. A Software Emulator which simulates the target military computer on the FASP host computer is provided. Unit tests of software modules can be performed at earlier stages of the development or maintenance process in the simulated environment in which it will operate. The cost of software errors are reduced by locating and correcting them earlier. Testing is also able to begin well before the implementation phase. An Automatic Test Analyzer determines which paths through the project program have been traversed and instruments the source code without hindering performance. Results are automatically stored in the FASP database. From there management reports on path tests are generated. FASP also supports Automatic Regression Testing. All module test results are maintained in the FASP database. Each module has a complete test history available. A change to a module will automatically retest all test cases affected by the module change and store the results to the database.

Facilities for implementation of FASP are extensive. The host system consists of two CDC 6600 and two CDC CYBER 175 computers. This large capacity system enables several projects to be maintained by FASP concurrently. Each programmer is able to use a virtual target machine emulated by the FASP host computers. Many virtual machines are able to be utilized concurrently. By combining the support of many projects on one system, significant physical plant cost savings are realized. The large computing capacity is also

available to handle urgent maintenance deadlines without significantly reducing normal development and maintenance activities.

FASP allows the use of nearly any computer to tie into its facilities. Contractors not physically located at the FASP sight are able to utilize FASP during the development phase of the project software. As the maintenance of the software is turned over to NADC, a smooth transition from the development phase to the maintenance phase is insured. The maintenance activity has available extensive documentation from development to aid maintenance.

FASP is the first attempt at an integrated software development and maintenance environment directed at embedded real time computer systems. It has met with considerable success on the three flight software systems developed and maintained on FASP. Its success is based on the facility being used throughout the entire software lifecycle. FASP would not be particularly suitable for use in systems in late maintenance phases such as A-6 or A-7. These older OFPs would require extensive rewrites and documentation before the FASP system could be utilized. FASP is best suited to be used from the initial development and throughout the remainder of the software lifecycle.

V. IMPROVING OFP MAINTENANCE THROUGH DOCUMENTATION

A. INTRODUCTION

Thus far this thesis has covered the background material to understanding the unique problems related to software maintenance of real time, embedded aviation software systems. Definitions have been presented and models compared. FASP, an environment in use by one software activity has been presented. The next two chapters of the thesis presents tools and methodologies which can be used to improve the maintenance effort with modest expenditures in time and money. Focus is directed in the two areas where the most improvement in the maintenance effort seems possible, documentation and testing. Both of these subjects were brought up time after time in dicussions with OFP maintenance personnel. It is likely when improvements in these areas are adopted, other areas of the problem will show improvement also.

The suggestions presented in the next two chapters by no means offer a quick easy solution. The problem has developed over a number of years and is much too complex. Solutions will not come easy no matter what price is paid. What follows is primarily based on interviews with the maintenance personnel conducted in an informal manner during the three trips to the two West Coast Navy flight software activities, during conferences and over the telephone.

B. DOCUMENTATION IMPROVEMENTS

Every software activity, every person involved in OFP maintenance mentioned one aspect of the OFP software to be severely lacking. That area is documentation. In nearly

every case the documentation the maintenance personnel received from the prime contractor when OFPs were turned over to Navy was poor. Many of the OFP contracts were written before any guidance from the Navy was available on documentation. In some flight systems the requirement for documentation was left out in order to save initial development funds. The already extremely difficult task of maintaining complex OFPs is made nearly impossible by the lack of good documentation.

Because of poor documentation many of the other problems of maintaining the OFP software develop. Training new personnel is made even harder as it takes a great deal of time for someone to understand a system in which there is poor reference material. The new personnel find themselves learning primarily through hands on experience. They learn the program on the fly as they implement changes. This slows the maintenance effort and affords an opportunity to introduce errors into the revised code.

Poor documentation causes the entire update cycle of the OFP to be longer than would be needed with proper documentation. Experienced personnel find themselves spending a significant amount of time merely trying to understand the existing OFP code prior to designing a maintenance change. Because of the effort required to comprehend the existing OFP, the largest portion of time spent in the maintenance cycle is spent in the design phase.

Lack of documentation currently inhibits most maintenance teams from using many off-the-shelf software tools and methodologies. Several personnel interviewed named tools that would help their effort significantly but were unable to use due to the difficulty involved in setting up the OFP documentation to allow the tools use. Most tools are designed to be used with a well documented product. Because of this, most of the tools used are developed internally and

are not able to be shared between different projects. This has lead the support systems and methods used by the different OFP projects becoming increasingly disjoint over the years. Each has become its own seperate enitity. This can partly be blamed on poor documentation.

Difficulty in understanding the code when designing a maintenance change leads to difficulty in determining meaningful program test requirements for the revised code. A poorly designed change due to a lack of understanding of what the program is suppose to do leads to greater testing costs. The number of design errors would decrease if the design engineer and programmer had quality documentation available.

Suggestions of what to do first center on a definition of documentation. Documentation is defined in a broad sense as the method of giving information about a computer program or system so that a reasonably trained person is able to understand the system, use it and modify it to fullfill new objectives. This definition is a modified version of one presented by Edmund Berkely [Ref. 18]. The point taken from this definition is documentation allows the person utilizing it to understand the system.

There are many arguments as to the most effective format of documentation. It is an area of computer science that is still under extensive study and interfaces directly with study of the human learning process. This thesis will offer no profound insights into the proper format of documentation. It will accept only the premise that workable documentation is critical to the maintenance of OFPs.

Again citing an active OFP maintenance effort, the A-6E, OFP documentation received from Grumman was felt to be totally inadequate for the task. It has several major problems which limit its use. First, it consists only of the OFP program listing and a set of math flow diagrams. The

math flow diagrams consist roughly in the format of crude flowcharts containing mathematical representations of what is occuring in the program code. They are very difficult to read for someone not intimately familar with them. Each flow symbol contains a large array of cryptic symbols which are difficult to follow. The math flows exist on paper and have been copied so many times that individual symbols are faded and extremely difficult to identify. Changes to a program involves converting the representation of the change into the math flow format, redrawing by hand the pages affected and inserting the change into the hard paper copy. This leaves significant opportunties for error. As the documentation stands it is totally inadequate for training an engineer or programmer. It also does not allow use of a set of capable software tools in a meaningful maintenance environment.

Two very important forms of documentation are missing from the A-6 OFP inventory. Current Software Requirements and Aircraft Performance Specification Documents are not available. The maintenance programmer cannot determine exactly what the program is suppose to do prior to attempting to glean how the OFP actually does it. A software redesign is significantly slowed by the effort required to understand the current program. Errors are introduced only because the redesigned code is not what the maintenance change called for. Aircraft performance requirements are equally important to determine the parameters involved in a redesign effort. They are also important in understanding the OFP code itself. Accurate information on what the aircraft is doing during certain phases of operation helps tell the engineer what is happening inside of the OFP. For example, what the program expects to see from a certain sensor at a certain time is determined from the aircraft performance specifications document.

The suggestions for improving the documentation do not involve a complete OFP rewrite. The documentation improvements will center around the OFP as it currently stands. These suggestions are aimed at improving the maintenance effort without very large expenditures of resources.

1. Electronic Documentation Storage

The first effort at improvement of the documentation would be to change the format on which it is kept. Automating the storage, retrieval and reducing the time involved to record a change would help greatly. The chance of not incorporating a change to one of the the many copies of the paper documentation is reduced. Electronic storage also allows the programmer to quickly retrieve the documentation he requires.

The A-6 software personnel are taking steps in exactly this direction. A Documentation Librarian has been hired to deal with the paper documentation and enter it into an electronic storage facility. One person or group of persons assigned only to the maintenance of the documentation will allow closer control of the accuracy of that documentation. The programmer will not need to burden himself with the requirement to enter changes to the documentation of code he has revised.

Many tools abound which would allow the documentation to be stored electronically. A database could be implemented on existing facilities or maintained on an expanded small network of microcomputers. The exact tools used are not as important as the concept of maintaining the documentation electronically to allow easy access, modification and storage of large amounts of data.

Characteristics of a system chosen should reflect the following:

1. Ease of Use. The system should be easy to use so as not to discourage the user. Ideally the system would be incorporated online within the system that the programmer normally works, as FASP does. Realistically a stand alone machine which does not require excessive effort to use is adequate.

2. Speed of Access. The system need not be such that instanteous access is achieved. Most microcomputer database systems with adequate memory allow the user to access text and print it out without a long wait. The number of personnel using the system is not large enough that concurrent multiple users are a significant problem.

3. Adequate Backup. This may seem incredibly obvious but it must be addressed if the documentation is to be stored in an electronic form. A system employed on a mainframe may utilize the operating system backup procedures normally used. Smaller microsystems would require multiple copies be maintained on tape and a procedure to insure timely backups implemented.

4. Consistency. Related to the backup question, consistency involves insuring that all copies of the data are consistent with each other. This must be accomplished if the electronic documentation is to be meaningful. Again, the exact system employed to hold the documentation would yield the method used to maintain consistency. As the database to contain the current documentation would not be extremely large the system to maintain consistency need not be automated.

5. Graphical Representation Capability. Conversion of the current documentation would involve working with paper copy which contains many graphical representations and symbols. The documentation should not require major redefinition or restructuring if the cost of maintaining it in an electronic form is to be held to reasonable level. The difficulty to use certain symbols contained in the math flow diagrams has slowed the effort at entering the A-6 documentation into the Xerox Star system that they are using. Switching from graphical to text modes is constantly required to properly position the symbols required by the math flow documentation.

The graphical representataion problem may be the key to selecting the documentation storage system. A careful survey should be conducted of the data to be entered into the new system prior to selecting the storage system to be used. The system selected should be able to easily represent any symbol required in the documentation. Some symbols contained in the current documentation may be able to be changed to allow the use of a particular storage system. This problem may limit the ability to use some of the microcomputer databases available. It calls for careful study to insure the stored data is accurate and that new data can be entered without excessive effort.

Costs to implement the electronic storage of the documentation vary with the volume to be stored and its current hard copy format. Rough estimates of the cost to purchase a capable microcomputer system with adequate hard disk storage, three to four terminals, adequate graphical representation capabilities, backup facilities, software (purchased if possible) and printers range from 20,000 up to 50,000 dollars. This figure represents a very small investment. Once implemented it could provide significant savings in the years to come. Cost for more extensive database systems to be used on larger computer facilities range higher. The cost of implementing the Xerox Star system for storage of A-6 documentation will run approximately 100,000 dollars. The personnel assigned to use the documentation feel that this money is well spent. While input of the current documentation is painful, the payoff in the long run will be well worth the initial investment.

After the system to store and manipulate the documentation has been implemented, the next step is to enter the documentation required by the lifecycle chart that figure 3.3 outlined. This would of course involve adapting the lifecycle methodology illustrated in figure 3.3 as a standard throughout the maintenance process. The documentation required at each step is then formated to be entered in the storage system after the completion of each phase. A documentation history is maintained for each maintenance change. The format is fixed and once the maintenance personnel become familar with it, lifecycle phase documentation generation and use will become easier. The system should have enough capacity so that change documentation can be stored online between OFP updates to allow easy access if required after completion of an update. When the next OFP update cycle is started, the documentation is also available to help comprehend the current state of the OFP. Once a new

update cycle is completed the previous update documentation is stored in an archival storage system for program history purposes. This would enable all change documentation to be maintained in order to be able to trace the OFP change history.

2. Software Requirement Document

The next step in improving the documentation of OFPs would be the generation of a Software Requirements Document (SRD). Work done by the Naval Research Laboratory [Ref. 1], yielded a workable Software Requirements Document for the A-7 OFP. This document was generated in preparation for recoding of the OFP. The generation of such a document for other existing flight systems need not entail recoding the OFP. The following outline of the format of the Software Requirements Document was modeled from the A-7 work done for the Naval Research Laboratory.

The primary purpose of the Software Requirements Document is to describe the externally visible behavior of the OFP without desribing the implementation. The SRD assumes the hardware to be static; this is a valid assumption for embedded aircraft systems. Interface characteristics are seperated from software requirements. Interface characteristics will change only if the hardware changes, while software requirements will change only if the mission requirement of the OFP changes. The document is maintained as the reference for what the aircraft OFP does. Implementation is not addressed. As an example of what might be contained in the document, it might explain that during final approach to an aircraft carrier landing, certain symbols are displayed on the pilot's heads up display (HUD), as opposed to the symbols that are displayed during a bombing run. How the computations occur that determine where to place the symbols on the HUD is not addressed.

As was done by the NRL work, the format is set up to enhance readability and is easily referenced. Tables are used extensively to make look up of specific items easy and enable the reader to easily spot missing data. To allow table usage, a standard set of definitions which represent long phrases or complex conditions are given symbols. They are referenced in a data dictionary contained within the document.

The format of the Software Requirements Document is discussed next, it was based on the design of the A-7 Software Requirements document.

a. Aircraft Computer

The A-7 Software Requirements Document begins with a short discussion of the aircraft's computer. This would be included in any other Software Requirements Document generated internally. The distingushing characteristics of the aircraft processor are highlighted. This section should be written with the newly arrived personnel in mind as a primary introduction into the aircraft computer system. Detailed descriptions of the computer are currently available and do not need to be included here.

b. Input and Output Data Items

The purpose of this section is to describe the interface between the aircraft processor and the devices which input and receive data from the processor. Input and output data are decribed as Data Items. This is the only section of the document which contains any information about the physical representation of the data. In follow on sections of the document, the Data Items and the values which they transmit are represented by symbolic names. Each Data Item is described in the following manner:

1. Each Data Item is given a symbolic name which is standardized throughout the document.

2. A prose description of its meaning and its relation to the device which utilizes it is given.

3. Numeric Data Items are typed as to accuracy and range requirements. Nonnumeric Data Items are given the mnemomic names of all possible values which may be assigned to it.

4. The format of the data representation is given.

5. The processor instruction sequence which is required to manipluate the Data Item (Read, Transmit, Write, etc) is described.

c. Operation Mode

Possible states of the program are defined in accordance with aircraft operating scenarios. A precise frozen scenario for a particular flight profile is described. It is very difficult to describe the state of the OFP at any given moment without a precise definition of what the aircraft is doing at the moment. This concept will be shown to critical in describing meaningful OFP tests.

When possible, modes described by available documentation should be used. If unavailable, the modes are described to match frequently encountered aircraft operating conditions. NRL chose five modes to describe: alignment, navigation, navigation update, weapon delivery and testing. The OFP is able to exist in more than one mode at a time. Exclusionary sets are defined to prevent combination of modes which are nonsense. The defintion of the operational modes should be done in cooperation of the OFP maintenance personnel, program simulator personnel, system design contractor and a group of experienced fleet users. Once a mode is defined and aggreed on, it is set and cannot be changed without agreement of the group described above. The definition on the mode would then have to be a careful process.

Events in aircraft operation which would cause the system to switch modes are also described in this section. An example would be an event which occurs in flight causing the mode to switch from navigation to a navigational update. This data is represented in table form. Conditions about the state of the program which are defined as true for a particular mode are also given in tabular form. These conditions are key to understanding the state of the program during a given mode.

d. Functions

The computation of Output Data Items is described as one of the many functions that the OFP performs. There are many functions involved during an executing OFP. Relations between the state of Input Data Items and the aircraft operating modes are provided. These relations allow the user to determine what coditions of the operating mode caused an Output Data Item to be produced. The operational modes selected in the above section are used. No reference is made to clock time.

e. Timing

The timing requirements for each function is stated in this section. An example would be the timing requirements of the updates for each display to the aircrew. The maximum delay between a request for an Input Data Item and the completion of processing yielding the Output Data Item is given. If understood, the system reaction to exceeding this value should be described. This section will be very difficult to complete. In some OFPs, such as A-6, the timing requirements between cycles is not static. Bounds would have to computed instead of finite values. The accurate completion of this section, while difficult, would be very valuable for future maintenance. Timing

considerations are poorly defined and difficult to deal with. They are critically important to the accurate operation of the OFP. An ability to reference the timing considerations for each OFP function could in fact be the most important aspect of the Software Requirements Document.

f. Accuracy

The accuracy requirements for the computation of all Output Data Items are given. This is another difficult and important part of the document. The first version of the A-7 Software Requirements Document did not have all of the data required to complete this section. It is a common complaint of maintenance personnel that they do not know the accuracy requirements of the data produced during computation by the OFP. Ground and air testing of the OFP may find that due to the lack of accuracy information that a function delivers incorrect Output Data Items causing the OFP to perform incorrectly. An example could as drastic as a weapon missing a target or the system crashing on uncomputable Input Data Items.

g. Undesired Events

Undesired events, such as processing an incorrect Input Data Item, elicit certain behavior from the OFP. This behavior is described. The entrance into an undesired situation should be from a standard aircraft operational mode as described earlier. Input of aircrews should be sought to determine the best response, or at least most commonly observed response to degraded OFP operation. This section could quickly grow in size and complexity if every combination of device and system failure is considered. A bound is set on this section by considering failure of the most important functions of the OFP, determined from user input and the predefined modes of OFP operation.

h. Partitioning

The allowable partitions of the OFP are described. Services which are computed by the OFP but are not mandatory for aircraft operation or execution of the OFP are described. Functions which may be canidates for removal at a future time are identified. This is an important aspect of the document in that the memory of the flight system is more often than not, limited. Incorporation of significant system enhancements may require the dropping of nonrequired functions. This would give the system engineers an easy reference to functions not needed.

i. Glossary

The glossary defines symbol names and acronyms of technical terms used throughout the document.

j. References

References used to gather the data contained in the SRD and Aircraft Technical Manuals are listed.

k. Data Dictionary

The standard terms used in function and Data Item description are listed and defined.

l. Index

The document is indexed in the following manner: By Data Item description, Mode description and usage and function Output Data Item.

The Software Requirements Document is not an easy quick fix to a documentation problem that has existed in some OFPs for years. It would not be cheap to implement. Perhaps it can be said that it does not fall within the scope of this thesis and offer a solution for the relative

short term. Consideration of the expected lifespan of the aircraft and the OFP itself must first be weighed prior to expending funds for development of such a document. If considered against the long expected lifespan of nearly every OFP, development of a workable Software Requirements Document is feasible. Before recoding of the OFP could be considered, a SRD would have to be written. Generating a SRD yields a document which is useful for current maintenance work and would be required for possible future OFP rewrites. The finished A-7 Software Requirements Document while extensive, is a workable document and appears easy to follow for someone familar with the terminology of the software it covers. Its format has been expressed by maintenance personnel as suitable for any OFP.

Generation of a good document for OFPs which currently lack one would be costly in time and money. In the cases where only a few personnel are familar with the entire OFP, their input into the document would be critical. It is also obvious that these personnel could not be expected to be utilized full time on the generation of the SRD without imparing ongoing OFP maintenance. The effort to write the document would have to be extended over a period of two to three years. The author feels this is not an execessive period of time. It covers approximately the production of two OFP updates. A training program could also be implemented around the SRD production in which all system maintenance personnel are involved. Familarity with the function of the OFP would be increased as the document was produced. Ultimately the document should be entered, stored and maintained on the electronic document storage system selected in the first step of documentation improvement.

3. Aircraft Performance Specification Document

The A-6 personnel complained about the lack of a document which outlined the performance of the aircraft itself. A document similar in function to the Software Requirements Document for the aircraft is needed in many projects. The generation of such a document would not have to involve maintenance personnel. It should be contracted out to the manufacturer and produced in a format suitable to the maintenance personnel. It also would be very useful as a training aid to help the new engineer understand the system which he will soon work. If maintained properly it would supply the maintenance personnel with an accurate definition of the performance profiles of the aircraft which directly affect the execution of the OFP. A general format is proposed.

a. General Description

A general description of the aircraft and detailed description of the mission definition are contained in the first section. This section merely provides an introduction to the platform and a starting point for understanding the rest of the system.

b. Mission Profiles

Mission profiles are defined next. They should, as close as possible, match the mode scenarios presented in the SRD. Normal values for the various devices which interface with the flight computer system are contained in tabular form. Tables are generated for each flight profile. The flight profiles again will impact greatly on the later testing phases of revised OFPs. To insure the generation of meaningful test data, the flight profiles are standardized. Extreme conditions which are faced under combat situations are also represented.

c. Degraded Operation

Expected degradations to the aircraft performance are outlined in this section. Battle damage to the aircraft which does not render the aircraft unflyable are given. The flight software personnel are able to determine the expected reduction in device demand and input to the OFP This section should be modeled closely after the Undesired Events chapter of the SRD.

d. Operating Ranges

Actual specification of the aircraft operating parameters are listed. Ranges of possible input and output values for the avionics which interfaces with the OFP are given. The devices are divided into aircraft subsystems such as navigation, HARPOON missile fire control and the like. This section serves as a quick reference to the actual values of the Input and Output Data Items used in the Software Requirements Document.

The Aircraft Performance Specification Document is not nearly as important to the programmer as it is to the system engineer. It may in fact allow the programmer to cross the boundry between programmer and engineer. It is usually available in some form from the manufacturer without a great deal of expense being involved. Money should not be spent on its development over the Software Requirements Document. It is a supplement to the SRD to be developed in parallel or after completion of the SRD.

VI. OFP TESTING IMPROVEMENTS

A. INTRODUCTION

The very large subject of OFP testing is addressed next. Testing of software is not an exact science by any means. Debate persists on methods of performing tests to yield meaningful and accurate results. Complex software, such as the OFPs, present even more difficult questions as to the best testing method. The complexity of the OFP presents the engineer with a software product that is basically in an untestable form. The state of the OFP is difficult to track. Because of this, it is very difficult to identify what conditions triggered a particular test result. Since the older OFPs are not modularized, code that requires modification is very difficult to isolate. How then can testing be structured to assure that meaningful results are attained? This is an extremely important question in consideration of the OFPs use in high performance aircraft. The engineer who certifies an OFP ready for live flight test must feel confident in his product. The complexity of the code must have been addressed during ground tests in order that operating conditions in the fleet will not trigger the OFP into a failure. The high reliability required of the OFP must be obtained during the test phase.

The testing portion of the aviation lifecycle has been identified as requiring the most resources to accomplish. Massive support facilities are required in the form of flight simulators. A great deal of support software is required to run simulations of the OFP. After ground tests, the OFP is tested in live flight tests. The live flight tests consume a large portion of money due to maintenance of the flight range facilities and aircraft fuel requirements.

Current test practices in most OFP maintenance facilities consist of running all or parts of the OFP on the target computer within the confines of a flight simulator. The simulator is usually written in a moderately high level programming lanuage such as Fortran. It is instrumented to attempt to track the state of the OFP during a test run. The simulator support facility is manned by different personnel than the actual OFP maintenance team. The simulator personnel write the support software that is used by the maintenance personnel to test the OFP.

B. WEAPON SYSTEM SUPPORT FACILITY

The simulators fall under the Weapon System Support Facility (WSSF) for each OFP. The WSSF is a total system by which OFP maintenance is supported to do testing. The WSSF serves three primary functions:

1. OFP validation and verification. Does the program work properly and did changes affect the remainder of the unchanged code?

2. New weapon integration. Design of new weapon interfaces with the entire flight software system.

3. Weapon system analysis. Measurement of simulated results of flight tests and weapon delivery scenarios.

The WSSF should be structured in an evolving state to be constantly improving the support of the OFP. The author reviewed the doctrines for the production of support software for the A-7, A-4, AV-8 and F-18 OFP projects. All were found to be well structured methodologies which conform to modern software engineering principles and techniques.

The subject of OFP testing can be seen to be viewed from two persectives. First the viewpoint of the maintenance personnel who need to test the integration of revised code into the entire OFP. The other belongs to the personnel assigned to develop and maintain the support facilities. The concept of what constitutes a successful test may not be

the same for each group. One manager of a support facility complained the maintenance personnel assigned to the OFP which he supported viewed simulator flight testing as a "stick and rudder affair." Meaning that the maintenance personnel were happy to load the OFP into the test facility and execute the OFP in accordance to a poorly defined model of the OFP during flight. The test lacked a specific structure. He felt this was a misguided approach to obtain a meaningful test of the OFP software.

The WSSF can be thought of as residing between the test requirements and the target flight computer system. The role of the WSSF centers around the data supplied to the target flight computer system during a test.

Since the focus of the WSSF is the supply of data to the target flight computer, OFP tests must be designed so that specific data is supplied to achieve a specific test result which is repeatable. The specific input data can be seen to bound the test process. The user and the WSSF should aggree on the bounds of the simulation and freeze it from frequent changes. The WSSF personnel are able to break the process of test procedure software generation into manageable modules based on the concrete definition of the test requirements.

The test data design is approached in the following manner. The component to be tested is identified and isolated. What needs to be tested to validate this component is identified. Once determined, specific test scenarios are designed by the maintenance and WSSF personnel.

The improvement of OFP testing will be centered on the tools and methodologies of the WSSF. Development of the WSSF is an ongoing process which attempts to constantly increase its capability to support the OFP test effort.

C. STANDARD FLIGHT TEST SCENARIOS

The most important aspect of the generation of WSSF software which will support OFP testing in a meaningful manner, hinges on standardized flight scenarios. The standardization of the scenarios attempts to let the maintenance personnel identify a flight profile which will exercise the OFP in such a way that realistic meaningful test results are obtained. A successful completion of the test flight scenario would yield flight data which is recorded into an output file. The data recorded is compared to the output data expected from this standard scenario. The output file and instrumentation data are stored for historical purposes.

The author witnessed the execution of two test flight scenarios run on the A-4 flight simulator. One scenario called for the aircraft to take off and execute a climb to 5,000 feet. From there it flew over a ground target. A dive was initiated and several turns were made around the target. All of this was represented by simulation of the HUD symbols on a CRT screen. The target was represented by a triangle which rotated on the screen in accordance to the movement of the aircraft. The symbols viewed on the screen were generated from the actual signals that the target computer generated as it executed the OFP. The WSSF input the data which lead the target computer to execute the OFP as if the aircraft were actually in flight performing the scenario described. The WSSF also provided the interface between the target computer and the CRT which allowed the HUD simulation. Instrumentation of the input to the aircraft avionics from the target computer is recorded by the WSSF into an output data file. Timing references are recorded to be able to compare the input data with the output data. The state of the OFP can hopefully be obtained and reasons why specific output data was generated determined.

The methodology of freezing the test scenarios is critical to the WSSF support software design process. The WSSF personnel and the maintenance personnel together decide on the scenarios which exercise various functions of the OFP. The freezing of the scenario definitions allow the WSSF personnel to implement a design process which is modular and can be automated to increase productivity. Scenarios are continually built which eventually enable the OFP to be exercised in such a way that the tested OFP can leave the software facility with a very high level of confidence.

D. WSSF PRODUCTION TOOLS

The increasing capability of the WSSF is critical to the reduction of the maintenance effort of the OFP. The production of WSSF software should be automated as much as feasible to help provide this increasing capability. The WSSF software does not face the storage, hardware support and timing constraints that must be considered of the OFP itself. The code generated can comply with up to date software engineering principles and use automation when possible.

Automation of the generation of WSSF code also has further advantages. The code produced initially is more likely to be considered correct. Routine repetitive tasks are eliminated, thus increasing productivity. The verification of the simulator code integrity is made simplier. Documentation of the simulator code can be made automatic. Analysis tools can be built for the test results. Portability can be obtained by using standard automation techniques.

The following software tools, based on the A-4/AV-8 WSSF development strategy, are designed around automating the production of WSSF software and automating the execution of

test scenarios. There are five tools mentioned which are explained below. The process starts with SREM and continues with the Module Generator and FLECS to actually produce module code. All three tools work in conjunction to produce the module code for software used in the flight simulator. The module was defined from the standardized flight scenarios covered earlier. AVSIM uses the input data for a particular test scenario to execute the modules, required to run that test scenario. AVDOC uses data from AVSIM and the Module Generator to produce standard forms for documentation of a test execution. The first three tools mentioned deal with WSSF software module production. The final two tools deal with helping to automate the test execution using the modules written by the first three tools.

1. SREM

SREM, Software Requirements Engineering Methodology, designed by TRW, is a tool which ties requirements to applications. A Requirement Statement Language (RSL) is used by SREM to generate input into the Module Generator (MOG). A module is first defined by stating the requirements of the module in the SREM RSL. The module definition in the RSL is processed and the results are input directly into the Module Generator. SREM is written in Pascal and utilizes a relational database. The database has the advantage of being able to be used with other applications other than SREM. SREM is the first tool in automating the production of WSSF software.

2. Module Generator

The Module Generator takes the output of SREM and acts as a FLECS-preprocessor producing FLECS code. FLECS, as will be seen, actually produces the module source code. MOG structures the application of modular code. MOG

addresses only the module input and output. A central dictionary is used to define module input and output. MOG also automatically inserts code into the module which is used by another tool to trace, debug and time the module.

3. FLECS

Fortran Language with Extended Control Structures (FLECS) is a tool which acts as a Fortran pre-processor generating Fortran 66 code. It has the capability to be extended to generate Fortran 77 control statements. It takes FLECS code from the MOG as its input and converts it into valid Fortran source code. It is a stand alone tool not tying directly to the MOG. This is the tool which yields the portability of the WSSF code. Currently all support facilities but one at NWC utilize Fortran 66. Code generated by FLECS is tranportable between the facilities. FLECS is the last major tool used in the generation of WSSF software. The next two tools deal with the execution of a test scenario using the software produced by the first three tools.

4. AVSIM

AVSIM, Avionics Simulation, provides an interface between the avionics hardware and the WSSF computer software. It controls the debug, trace and timing options of the WSSF code generated by the MOG. The tool is able to configure itself in accordance with the data contained in the input data file for the test. It is able to turn on the WSSF modules required to run a test of the OFP by analysis of the test input data file. AVSIM runs the simulation. This is an important automatic feature of the test facility. Tests are much easier to set up and run. Once input data for a particular test is standardized, the output data expected by running of the modules AVSIM turns on can be standardized also.

5. AVDOC

AVDOC, AVSIM Documentation, generates predefined forms from the module generator source files. These forms include: status reports, symbol dictionary listings, cross reference guides and keyword searches of the modules turned on by AVSIM in the execution of a test scenario. It is able to tie directly with the AVSIM and MOG tools. AVDOC can be thought of as a book keeping program that expands AVSIM information to produce predefined documentation forms.

6. Example

After a standardized flight scenario is defined by the OFP maintenance and WSSF personnel, it is broken into modules. The WSSF personnel take each module and define it in terms of its requirements in the SREM Requirements Statement Language. After this is processed, the Module Generator structures the input and output of the module in FLECS code. MOG also adds code which is used by the simulation tool, AVSIM, to trace, debug and time the module. The tool FLECS takes the output of the MOG and produces valid Fortran source code for that module. The generation of the module is completed. AVSIM is used to execute the required modules for a particular flight test scenario automatically. Which modules are activated are based on the input data file for the simulated flight test. Once the first three tools generate the module code, the module may be stored until activated by AVSIM. AVSIM also executes the timing, debugging and trace code during the test execution. AVDOC is activated when a test is run to produce standard forms, documenting the test execution.

7. WSSF Tool Summary

These are the primary tools used to automate software production and use at one WSSF at NWC. This methodology to establish an increasing capability in WSSF development impressed the author enough that it was felt that a similar approach should be taken on all OFP test facilities. Exact tools used will depend on the computer resources available. The notion of fixed flight scenarios worked out between the simulator and maintenance personnel should be adopted. Until the OFPs are structured properly, the biggest payoff in increasing the ability to meaningfully test the OFP resides in the WSSF development.

VII. CONCLUSIONS

A. CONCLUSIONS

The thesis concludes with several observations and recommendations concerning the development and maintenance of future flight software systems.

1. Design It Right

Future OFPs must learn from the mistakes made during the development of nearly every current operational OFP. While it is not always necessary to design the flight software in a high level language, structuring the code into modules is crtical to keeping maintenance costs down. The hardware system employed should be designed with enough memory and processor capability to handle the first versions of the OFP and expected updates without severe loss of code structure during optimization. Documentation should be produced in accordance with current guidelines. The production of a well designed Software Requirements Document is the minimal acceptable documentation. Methods for defining interfaces for code developed by different sources need to be defined. As aircraft computer systems become more complex, single contractor developed OFPs will become rare. The interfaces will prevent integration nightmares when the final product is brought together.

2. Development/Maintenance Environments

Work should continue on environments such as FASP. New environments need to be defined to support software for future flight systems. Navy flight software activities should be allocated funds now to begin development of a

common development/maintenance environment to be used on all flight software. These general purpose environments would be defined within guidelines that contractors must follow during OFP development. When the Navy flight software activities assume maintenance responsibility, the change will be smooth and maintenance easier. This recommendation will not be cheap to implement, but if the quality of future flight software systems is to remain high the money should be spent now.

3. Money

More funds should be allocated now to improve the maintenance of operational flight software. As this thesis hoped to propose, great expenditures on the current flight software need not be made. When considered against the cost of a single aircraft it seems incredible that flight software activities must spend operating funds to develop a very badly needed Software Requirements Document. The generation of a SRD is not expensive when the savings it will generate over the remainer of the OFP lifecycle are recognized.

4. Education

Two recommendations concerning education are made. The first centers around the personnel who make decisions on flight software contracts and fund allocation. From the observations made by the author during research for this thesis, it was felt that past high level administrators of flight software funds and contracts knew nothing about software at all. One manager of an OFP maintenance team relayed the story of a high level administrator located well above the trenches asking him how much did the flight software for a particular aircraft weigh. He needed to know this in order to allocate funds concerning that software. When this type of knowledge level is faced from those who control

funds for software development and maintenance it is easy to see why some of the errors made in the past were committed. Personnel who understand the nature of real time embedded software and the general principles of software engineering should be placed in more responsible positions.

The second aspect of education revolves around training new engineers for employment in the flight software laboratory. No facilities exist for training an engineer on a system such as A-6 or F-14. The Navy should begin a program where engineers are identified in the academic institutions, sponsored and trained in engineering and computer courses which would most help in working on flight computer systems. This person would then obligate to work for the Navy for a minimum length of time.

B. FINAL CONCLUSIONS

All of the recommendations made in this thesis present difficult decisions concerning expenditures of funds for aviation flight software maintenance. There are those who feel that the expenditure of these funds are not needed. The fact remains that if the high quality of Naval flight software is to continue, these difficult decisions must be made and the money spent.

LIST OF REFERENCES

1. Heninger, K.L. and Kallander, J.W. and Shore, J.E. and Parnas, D.L. Software Requirements Document for the A-7E Aircraft, pp 1-12, Naval Research Laboratory Memorandum Report 3876, Naval Research Laboratory, Washington, D. C., November 1978

2. Lientz, B.P. and Swanson E.B., Software Maintenance Management, pp 67-79, Addison-Wesley Publishing Company, 1980

3. Hamlen, W.T. and Fjeldstad E.F., Application Program Maintenance Study, Tutorial on Software Maintenance, pp 13-31, IEEE Computer Society Press, 1983

4. Bristow, G., Tools to Aid the Specification and Design of Flight Software, pp 2-18, Department of Computer Science, University of Colorado, Boulder Colorado, January 1980

5. Van Horn, E.C., Software Must Evolve, Tutorial on Software Maintenance, IEEE Computer Society Press, 1983

6. Howden, W., Contemporary Software Development Environments, Communications of the ACM, Volume 25, Number 5, pp 318-329, Association for Computing Machinery, 1982

7. Wasserman, A.I., Guest Editor's Introduction: Automated Development Environments, pp 7-11, Computer, Volume 14 Number 4, April 1981

8. Naval Air Development Center, Facility for Automated Software Production, Software Production and Maintenance Methodology, pp 1-25, Naval Air Development Center, Warminster, Pennsylvania, 1979

9. Boehm, B.W., Software Engineering Economics, pp 35-55, 641-690, Prentice-Hall, Inc, 1981

10. Parikh, G. and Zvegintov, N., Tutorial on Software Maintenance 1-11, IEEE Computer Society Press, 1983

11. National Bureau of Standards Computer Science and Technology, Features of Software Development Tools, NBS Pub 500-74 pp 1-6, National Bureau of Standards, Washington D. C., February 1981

12. Neetz, R.A., Human Factors in Software Design: A Problem Indentification, pp 1-46, Weapons Control and

Software Systems Division, Pacific Missile Test Center, Point Mugu, California, November 1981

13. Department of Defense, Software Technology for Adaptable Reliable Systems Program Strategy, pp 2-14, Department of Defense, March 1982

14. Scheidewind, N.F., Software Maintenance: Improvement Through Better Development Standards and Documentation, pp 28-30, Naval Postgraduate School, Monterey, California, February 1982

15. De Roze, B.C. and Nyman, T.H., The Software Lifecycle- A Management and Technological Challenge in the Department of Defense. IEEE Transactions of Software Engineering, Volume 4, No. 4, July 1978

16. Shooman, M.L., Software Engineering, Design, Reliability, Management, pp 14-20, Computer Science Series, McGraw-Hill, 1983

17. Nassi, I.R., A Critical Look at the Process of Tool Development: An Industrial Perspective, pp 41-51, Digital Equipment Corporation, Maynard Mass, 1983

18. Berkeley, E., Computer Assisted Documentation of Working Binary Computer Programs with Unknown Documentation, Proceedings of the Third Symposium on Computer and Informational Software Engineering Sciences, COINS III, Volume 2, Academic Press, 1971

BIBLIOGRAPHY

Brooks, R., Using Behavior Theory of Program Comprehension in Software Engineering, Tutorial on Software Maintenance, IEEE Computer Society Press, Los Angles, 1983

Corwell, W.R., and Osterweil, L.J., The Toolpack/IST Programming Environment, Aregonne National Laboratory, Aregonne Illinois, 1983

Gray, M. and Londcn, K.R., Documentation Standards, Brandon/Systems Press Inc. New York, 1969

Lientz, B.P. and Swanson, E.B. and Tomkins, G.E., Characteristics of Application Software Maintenance, Communications of the ACM, Volume 21, Number 6, Association for Computing Machinery, New York, 1978

Lientz, B.P. and Swanson, E.B., Problems in Application Software Maintenance, Communications of the ACM, Volume 24, Number 11, Association for Computing Machinery, New York, 1981

Naval Weapons Center, AV-8B Weapons System Support Facility, A-4/AV-8 Facility Branch, Naval Weapons Center, China Lake, California, September 1983

Naval Weapons Center, Software Development Plan and Documentation Standard, A-7 Facility Branch, Naval Weapons Center, China Lake, California, 1982

Naval Weapons Center, System Development Plan for the F/A-18 Weapons System Support Facility, Naval Weapons Center Aircraft Weapons Integration Department, China Lake, California, October 1983

Prokop, J., Computers in the Navy, Naval Institute Press, Annapolis, Maryland, 1976

Sohar Inc, Computer Science and Technology, The Introduction of Software Tools, National Bureau of Standards, Washington, D.C., 1982

Warnier, J.D., Reliabilty and Maintenance of Large Systems, Tutorial on Software Maintenance, IEEE Computer Society Press, Los Angles, 1983

Wasserman, A.I., Tutorial: Software Development Environments, IEEE Computer Society Press, New York, 1981 to no

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Camden Station Alexandria, Virginia 22314	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943	2
3.	Department Chairman, Code 52 Department of Computer Science Monterey, California 93943	1
4.	LT. Robert B. Upchurch 230 East Parkway Drive Columbia, Missouri 65201	2
5.	Dr. Gordon Bradley Code 52BZ Department of Computer Science Naval Postgraduate School Monterey, California 93943	4
6.	Capt Bradford D. Mercer, USAF, Code 52BI Department of Computer Science Naval Postgraduate School Monterey, California 93943	1
7.	Steve Underwood, Code 3192 A-6 Software Development Branch Head Naval Weapons Center China Lake, California 93555	1

Thesis
U458
c.1

Upchurch
Improvements to soft-
ware maintenance methods
in real time embedded
aviation flight systems.